aws

ユーザーガイド

Amazon ECR

API バージョン 2015-09-21

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

Amazon ECR: ユーザーガイド

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

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

Table of Contents

Amazon ECR とは 1
概念とコンポーネント 1
一般的なユースケース 3
Amazon ECR の機能 5
Amazon ECR の開始方法 6
Amazon ECR の料金表 7
ライフサイクルを通じたイメージの移行 8
前提条件 8
のインストール AWS CLI 8
Docker をインストールする 8
ステップ 1: Docker イメージを作成する 10
ステップ 2: リポジトリを作成する 12
ステップ 3: デフォルトレジストリに対して認証する 13
ステップ 4: イメージを Amazon ECR にプッシュする 13
ステップ 5: Amazon ECR からイメージをプルする 15
ステップ 6: イメージを削除する 15
ステップ 7: リポジトリを削除する 16
パフォーマンスの最適化 17
リクエストの実行 19
IPv6 の使用開始 19
IP アドレス互換性のテスト 20
デュアルスタックのエンドポイントを使用したリクエストの実行 21
Docker CLI から Amazon ECR エンドポイントを使用する 21
IAM ポリシーでの IPv6 アドレスの使用 22
プライベートレジストリ 24
レジストリの概念 24
レジストリの認証 24
Amazon ECR 認証情報ヘルパーを使用する 25
認可トークンを使用する 25
HTTP API 認証を使用する 26
レジストリ設定 27
Blob マウント 28
Blob マウントの概念 28
Blob マウント設定 29

レジストリ許可 30
レジストリポリシーの例 31
拡張レジストリポリシースコープに切り替える 34
クロスアカウントレプリケーション用のアクセス許可の付与 36
プルスルーキャッシュ用のアクセス許可の付与 38
プライベートリポジトリ 40
リポジトリの概念 40
イメージを保存するためのリポジトリの作成 41
次の手順 43
リポジトリの詳細の表示 43
リポジトリの削除 45
リポジトリポリシー 45
レポジトリポリシーと IAM ポリシー 46
リポジトリポリシーの例 48
リポジトリポリシーステートメントを設定する 53
リポジトリのタグ付け 55
タグの基本 55
請求用のリソースにタグを付ける 56
タグを追加する 56
タグの削除 58
プライベートイメージ 59
イメージのプッシュ 60
必要な IAM 許可 60
Docker イメージをプッシュする 62
マルチアーキテクチャイメージのプッシュ 64
Helm チャートをプッシュする 66
アーティファクトの削除 68
イメージの詳細を表示する 71
イメージのプル 72
Amazon Linux コンテナイメージのプル 73
イメージの削除 75
イメージのアーカイブ 77
ECR アーカイブストレージクラスとは 77
イメージのアーカイブ 78
イメージの復元 80
イメージにもう一度タグを付ける 81

イメージタグの上書きの防止 ... 84
イメージタグのミュータビリティの設定 (AWS マネジメントコンソール) ... 84
イメージタグのミュータビリティの設定 (AWS CLI) ... 85
コンテナイメージマニフェストの形式 ... 87
Amazon ECR イメージマニフェストの変換 ... 87
Amazon ECS での Amazon ECR イメージの使用 ... 88
必要な IAM 許可 ... 89
タスク定義での Amazon ECR イメージの指定 ... 90
Amazon EKS での Amazon ECR イメージの使用 ... 91
必要な IAM 許可 ... 91
Amazon EKS クラスターに Helm チャートをインストールする ... 92
イメージの署名 ... 95
署名方法を選択する ... 95
考慮事項 ... 95
マネージド署名 ... 96
前提条件 ... 96
開始方法 ... 97
考慮事項 ... 100
署名の検証 ... 100
Amazon EKS によるマネージド検証 ... 100
Amazon ECS 用の Lambda アドミッションコントローラー ... 101
Notation CLI による手動検証 ... 101
Notation クライアントの認証を設定する ... 101
手動署名 ... 101
前提条件 ... 102
イメージをスキャンして脆弱性がないか調べる ... 103
リポジトリ用のフィルター ... 104
フィルターのワイルドカード ... 104
拡張スキャン ... 105
拡張スキャンの考慮事項 ... 105
拡張スキャンの期間を変更する ... 106
必要な IAM 許可 ... 107
拡張スキャンの設定 ... 108
EventBridge イベント ... 110
結果の取得 ... 115
ベーシックスキャン ... 117

基本スキャンのオペレーティングシステムのサポート 117
基本スキャンの設定 118
イメージの手動スキャン 119
結果の取得 120
イメージスキャニングのトラブルシューティング 121
スキャンステータス SCAN_ELIGIBILITY_EXPIRED を理解する 122
アップストリームレジストリの同期 124
リポジトリ作成テンプレート 125
プルスルーキャッシュルールを使用するための考慮事項 125
必要な IAM 許可 127
レジストリ許可の使用 128
次の手順 130
クロスアカウント ECR から ECR PTC へのアクセス許可を設定する 130
クロスアカウント ECR から ECR へのプルスルーキャッシュに必要な IAM ポリシー 131
プルスルーキャッシュルールの作成 133
前提条件 134
の使用 AWS マネジメントコンソール 134
の使用 AWS CLI 143
次の手順 147
プルスルーキャッシュルールの検証 148
プルスルーキャッシュルールによるイメージのプル 149
アップストリームリポジトリの認証情報を保存する 151
リポジトリプレフィックスをカスタマイズする 161
プルスルーキャッシュに関する問題のトラブルシューティング 162
イメージのレプリケート 164
レプリケーションポリシー要件 164
ポリシー設定の概要 164
送信先レジストリポリシーの要件 165
ソースアカウント要件 166
一般的な誤解 166
レプリケーション失敗のトラブルシューティング 166
プライベートイメージのレプリケーションに関する考慮事項 167
レプリケーションの例 169
例: 単一の送信先リージョンへのクロスリージョンレプリケーションの設定 169
例: リポジトリフィルターを使用したクロスリージョンレプリケーションの設定 169
例: 複数の送信先リージョンへのクロスリージョンレプリケーションの設定 170

例: クロスアカウントレプリケーションの設定 ... 171
例: 1 つの設定内での複数のルールの指定 ... 171
例: すべてのレプリケーション設定を削除する ... 172
レプリケーションの設定 ... 172
レプリケーションの削除 ... 175
リポジトリ作成テンプレート ... 177
仕組み ... 177
リポジトリ作成テンプレートの作成 ... 181
リポジトリ作成テンプレートを作成するための IAM アクセス許可 ... 181
カスタムポリシーの作成 ... 182
IAM ロールを作成する ... 183
リポジトリ作成テンプレートの作成 ... 184
リポジトリ作成テンプレートの更新 ... 189
リポジトリ作成テンプレートの削除 ... 190
イメージのクリーンアップの自動化 ... 192
ライフサイクルポリシーの機能 ... 192
ライフサイクルポリシーの評価ルール ... 193
ライフサイクルポリシーのプレビューを作成する ... 195
ライフサイクルポリシーの作成 ... 197
前提条件 ... 197
ライフサイクルポリシーの例 ... 199
ライフサイクルポリシーのテンプレート ... 199
イメージの経過日数によるフィルタリング ... 200
最終プル時刻のフィルタリング ... 201
アーカイブ移行時間のフィルタリング ... 202
イメージ数によるフィルタリング ... 202
複数のルールによるフィルタリング ... 203
単一のルールで複数のタグをフィルタリングする ... 206
すべてのイメージでのフィルタリング ... 208
アーカイブの例 ... 211
ライフサイクルポリシーのプロパティ ... 213
ルールの優先順位 ... 213
説明 ... 214
タグステータス ... 214
タグパターンリスト ... 214
タグプレフィックスリスト ... 215

ストレージクラス 215
カウントタイプ 216
カウント単位 216
カウント数 216
Action 217
プルタイム更新の除外 218
プルタイム更新の除外の管理 218
プルタイム更新の除外に関する考慮事項 221
セキュリティ 222
Identity and Access Management 222
オーディエンス 223
アイデンティティを使用した認証 223
ポリシーを使用したアクセスの管理 224
Amazon Elastic Container Registry と IAM が連動するしくみ 226
アイデンティティベースのポリシーの例 231
タグベースのアクセスコントロールを使用する 235
AWS Amazon ECR の マネージドポリシー 237
サービスにリンクされたロールの使用 244
トラブルシューティング 254
データ保護 256
保管中の暗号化 257
コンプライアンス検証 265
インフラストラクチャセキュリティ 265
インターフェイス VPC エンドポイント (AWS PrivateLink) 266
サービス間での不分別な代理処理の防止 275
モニタリング 278
サービスクォータの可視化とアラームの設定 279
使用量メトリクス 280
使用状況レポート 282
リポジトリメトリクス 282
CloudWatch メトリクスの有効化 283
使用できるメトリクスとディメンション 283
CloudWatch を使用したメトリクスの表示 284
イベントと EventBridge 284
Amazon ECR のサンプルイベント 284
での AWS CloudTrailアクションのログ記録 292

CloudTrail の Amazon ECR 情報 ... 292
Amazon ECR ログファイルエントリの理解 ... 294
AWS SDKs の使用 ... 311
コードの例 ... 313
基本 ... 314
Hello Amazon ECR ... 314
基本を学ぶ ... 318
アクション ... 375
サービスクォータ ... 433
での Amazon ECR サービスクォータの管理 AWS マネジメントコンソール ... 440
API 使用量メトリクスをモニタリングするための CloudWatch アラームの作成 ... 441
トラブルシューティング ... 442
Docker のトラブルシューティング ... 442
Docker ログに予想されるエラーメッセージが含まれていない ... 442
Amazon ECR レポジトリからイメージをプルするときのエラー: "ファイルシステムの検証に失敗しました" または "404: イメージが見つかりません" ... 443
Amazon ECR からイメージをプルする際のエラー: "ファイルシステムレイヤーの検証に失敗しました" ... 444
レポジトリへのプッシュの際の HTTP 403 エラー、または "基本的な認証情報がありません" エラー ... 444
Amazon ECR エラーメッセージのトラブルシューティング ... 445
HTTP 429: リクエストが多すぎる、または ThrottleException ... 445
HTTP 403: "ユーザー [arn] は [operation] の実行を許可されていません" ... 446
HTTP 404: "レポジトリは存在しません" エラー ... 446
エラー: Cannot perform an interactive login from a non TTY device (TTY 以外のデバイスから対話型ログインを実行できません) ... 447
Amazon ECR での Podman の使用 ... 448
Amazon ECR での認証に対する Podman の使用 ... 448
Podman とともに Amazon ECR 認証情報ヘルパーを使用する ... 448
Podman を使用した Amazon ECR からのイメージのプル ... 449
Podman を使用した Amazon ECR のコンテナの実行 ... 449
Podman を使用した Amazon ECR へのイメージのプッシュ ... 449
ドキュメント履歴 ... 451
... cdlx

Amazon Elastic Container Registry とは

Amazon Elastic Container Registry (Amazon ECR) は、安全でスケーラブル、信頼性の高い AWS マネージドコンテナイメージレジストリサービスです。Amazon ECR は、IAM AWS を使用したリソースベースのアクセス許可を持つプライベートリポジトリをサポートします。これは、指定されたユーザーまたは Amazon EC2 インスタンスがコンテナリポジトリとイメージにアクセスできるようにするためです。任意の CLI を使用して、Docker イメージ、Open Container Initiative (OCI) イメージ、および OCI 互換アーティファクトをプッシュ、プル、管理することが可能です。

> **Note**
>
> Amazon ECR は、パブリックコンテナイメージリポジトリもサポートしています。詳細については、Amazon ECR Public ユーザーガイドの「Amazon Elastic Container レジストリとは」を参照してください。

AWS コンテナサービスチームは、GitHub でパブリックロードマップを維持します。これには、チームが取り組んでいる内容に関する情報が含まれており、すべての AWS 顧客が直接フィードバックを提供できるようにします。詳細については、AWS Containers Roadmap を参照してください。

Amazon ECR の概念とコンポーネント

Amazon ECR は、 AWSが提供するフルマネージド型の Docker コンテナレジストリサービスです。これを使用して、Docker コンテナイメージを安全かつ信頼できる方法で保存、管理、デプロイできます。これらの概念とコンポーネントは連携して、 内で安全でスケーラブルで信頼性の高い Docker コンテナレジストリサービスを提供し AWS、コンテナ化されたアプリケーションを効率的に管理およびデプロイできるようにします。

Amazon ECR の主要な概念とコンポーネントを以下に示します。

[Registry]

Amazon ECR レジストリは、各 AWS アカウントに提供されるプライベートリポジトリであり、1 つ以上のリポジトリを作成できます。これらのリポジトリを使用すると、Docker イメージ、Open Container Initiative (OCI) イメージ、およびその他の OCI 互換アーティファクトを AWS 環境内に保存および配布できます。詳細については、「Amazon ECR プライベートレジストリ」を参照してください。

認可トークン

クライアントがイメージをプッシュおよびプルするには、 AWS ユーザーとして Amazon ECR プライベートレジストリに対して認証する必要があります。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。

リポジトリ

Amazon ECR のリポジトリは、Docker イメージ、Open Container Initiative (OCI) イメージ、およびその他の OCI 互換アーティファクトを保存できる論理コレクションです。単一の Amazon ECR レジストリ内に複数のリポジトリを用意して、コンテナイメージを整理することができます。詳細については、「Amazon ECR プライベートリポジトリ」を参照してください。

リポジトリポリシー

リポジトリポリシーを使用して、リポジトリとリポジトリ内のコンテンツへのアクセス権を制御できます。詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。

イメージ

リポジトリには、コンテナイメージをプッシュおよびプルできます。開発システムでこれらのイメージは、ローカルに使用することや、Amazon ECS タスク定義と Amazon EKS ポッド仕様で使用することができます。詳細については、「Amazon ECS での Amazon ECR イメージの使用」および「Amazon EKS での Amazon ECR イメージの使用」を参照してください。

ライフサイクルポリシー

Amazon ECR ライフサイクルポリシーを使用すると、古いイメージや未使用イメージをプルーニングしたり期限切れにしたりするためのルールを定義することで、イメージのライフサイクルを管理できます。詳細については、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。

イメージスキャン

Amazon ECR には、コンテナイメージ内のソフトウェアの脆弱性を特定するのに役立つ統合イメージスキャン機能が備わっています。詳細については、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。

アクセスコントロール

Amazon ECR は、IAM を使用してリポジトリへのアクセスを制御します。Amazon ECR リポジトリをプッシュ、プル、または管理するための特定のアクセス許可を持つ IAM ユーザー、グループ、ロールを作成できます。詳細については、「Amazon Elastic Container Registry のセキュリティ」を参照してください。

クロスアカウントおよびクロスリージョンレプリケーション

Amazon ECR は、可用性の向上とレイテンシーの短縮のために、複数の AWS アカウントとリージョンにまたがるイメージのレプリケーションをサポートしています。詳細については、「Amazon ECR でのプライベートイメージレプリケーション」を参照してください。

暗号化

Amazon ECR は、 AWS KMSを使用した、保管中の Docker イメージのサーバー側の暗号化をサポートしています。詳細については、「Amazon ECR でのデータ保護」を参照してください。

AWS Command Line Interface Integration

AWS CLI には、イメージの作成、一覧表示、プッシュ、プルなど、Amazon ECR リポジトリとやり取りするためのコマンドが用意されています。

AWS マネジメントコンソール

Amazon ECR は を通じて管理することもでき AWS マネジメントコンソール、リポジトリとイメージを操作するための使いやすいウェブインターフェイスを提供します。

AWS CloudTrail

Amazon ECR は と統合されているため AWS CloudTrail、セキュリティとコンプライアンスの目的で Amazon ECR に対して行われた API コールを記録および監査できます。詳細については、「を使用した Amazon ECR アクションの口グ記録 AWS CloudTrail」を参照してください。

Amazon CloudWatch

Amazon ECR には、 Amazon CloudWatchを使用してモニタリングできるメトリクスとログが用意されており、Amazon ECR リポジトリのパフォーマンスと使用状況を追跡できます。詳細については、「Amazon ECR リポジトリメトリクス」を参照してください。

マネージド署名

マネージド署名は、イメージが Amazon ECR にプッシュされると暗号化署名を自動的に生成し、コンテナイメージの署名を簡素化します。詳細については、「マネージド署名」を参照してください。

Amazon ECR の一般的なユースケース

Amazon ECR は、 AWSが提供するフルマネージド型の Docker コンテナレジストリサービスです。Docker コンテナイメージを保存および配布するための安全でスケーラブルなリポジトリを提供

するため、コンテナ化されたアプリケーションデプロイに不可欠なコンポーネントです。Amazon ECR は、コンテナ化されたアプリケーションをさまざまな AWS サービスやオンプレミス環境で構築、配布、実行するプロセスを簡素化します。

Amazon ECR の主要なユースケースのいくつかを次に示します。

コンテナイメージの保存と配布

Amazon ECR は、Docker コンテナイメージを組織内で、または一般公開用に保存および配布するための一元化されたリポジトリとして機能します。開発者はコンテナイメージを Amazon ECR にプッシュし、Amazon EC2 AWS、 AWS Fargate Amazon EKS などの 内の任意のコンピューティング環境からプルできます。詳細については、「Amazon ECR プライベートリポジトリ」を参照してください。

継続的な統合と継続的なデプロイ (CI/CD)

Amazon ECR は AWS CodeBuild、、 AWS CodePipeline、およびその他の CI/CD ツールとシームレスに統合され、コンテナ化されたアプリケーションの自動構築、テスト、デプロイを可能にします。コンテナイメージは、CI/CD パイプラインの一部として Amazon ECR に自動的にプッシュできるため、さまざまな環境間で一貫性のある信頼性の高いデプロイが可能になります。

マイクロサービスアーキテクチャ

Amazon ECR は、アプリケーションがコンテナとしてパッケージ化された、より小さな分離されたサービスに分解されるマイクロサービスアーキテクチャに適しています。各マイクロサービスは独自のコンテナイメージを Amazon ECR に保存することができ、それによって個々のサービスの開発、デプロイ、およびスケーリングを独立して行うことが可能になります。

ハイブリッドおよびマルチクラウドデプロイ

Amazon ECR では、Docker Hub やサードパーティーレジストリなどの他のコンテナレジストリからコンテナイメージをプルする機能がサポートされています。これにより、組織はコンテナイメージの中央リポジトリとして Amazon ECR を使用することで、ハイブリッドまたはマルチクラウド環境全体で一貫したデプロイモデルを維持できます。

アクセスコントロールとセキュリティ

Amazon ECR にはきめ細かなアクセスコントロールメカニズムが用意されており、組織はレジストリに対してコンテナイメージをプッシュまたはプルできるユーザーを制御できます。また、認証と認可 AWS Identity and Access Management のために と統合され、コンテナイメージへの安全なアクセスを確保します。詳細については、「Amazon Elastic Container Registry のセキュリティ」を参照してください。

イメージの脆弱性スキャン

Amazon ECR には、ソフトウェアの脆弱性や潜在的な設定ミスがないかを調べる、コンテナイメージの自動スキャンが用意されており、安全でコンプライアンスに準拠したコンテナ環境を維持するのに役立ちます。詳細については、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。

プライベートコンテナレジストリ

セキュリティまたはコンプライアンス要件が厳しい組織では、Amazon ECR をプライベートコンテナレジストリとして使用して、機密性の高いコンテナイメージがパブリックレジストリに公開されず、組織の AWS 環境内でのみアクセスできるようにします。詳細については、「Amazon ECR プライベートレジストリ」を参照してください。

Amazon ECR レプリケーションを使用してグローバルに配布されるアプリケーションデプロイ

Amazon ECR レプリケーション機能を活用して、コンテナ化されたウェブアプリケーションイメージをプライマリリポジトリに一元化し、複数の AWS リージョンにまたがる自動分散を可能にし、世界中の低レイテンシーで一貫したグローバルデプロイを実現し、運用上の負担を軽減できます。詳細については、Amazon ECR でのプライベートイメージレプリケーションを参照してください。

古いコンテナイメージの自動クリーンアップ

Amazon ECR ライフサイクルポリシーを使用すると、経過時間、カウント、タグなどの定義されたルールに基づく古いコンテナイメージの自動クリーンアップが可能になり、ストレージコストの最適化、整理されたレジストリの維持、セキュリティとコンプライアンスの強化、自動化による開発ワークフローの合理化を実現できます。詳細については、Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化を参照してください。

Amazon ECR の機能

Amazon ECR には次の機能があります。

- ライフサイクルポリシーを使用すると、リポジトリ内のイメージのライフサイクルを管理できます。未使用のイメージをクリーンアップするルールを定義します。ルールはリポジトリに適用する前にテストできます。詳しくは、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。
- イメージスキャンは、コンテナイメージ内のソフトウェアの脆弱性を特定するのに役立ちます。各リポジトリはプッシュ時にスキャンするように設定できます。その場合、リポジトリにプッシュさ

れる新しい各イメージが確実にスキャンされます。その後、イメージスキャンの結果を取得できます。詳しくは、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。

- クロスリージョンおよびクロスアカウントレプリケーションを使用すると、必要な場所にイメージを簡単に作成できます。これは、レジストリ設定として、リージョンごとに構成されます。詳細については、「Amazon ECR でのプライベートレジストリ設定」を参照してください。
- プルスルーキャッシュルールは、プライベート Amazon ECR レジストリのアップストリームレジストリ内のリポジトリのキャッシュ方法を提供します。プルスルーキャッシュルールを使用して、Amazon ECR は定期的にアップストリームレジストリに問い合わせて、Amazon ECR プライベートレジストリにキャッシュされたイメージが最新であることを確認します。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリの同期」を参照してください。
- リポジトリ作成テンプレートを使用すると、プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に Amazon ECR がユーザーに代わって作成したリポジトリの設定を定義できます。タグのイミュータビリティ、暗号化設定、リポジトリポリシー、ライフサイクルポリシー、および自動的に作成されたリポジトリのリソースタグを指定できます。詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。
- マネージド署名は、イメージが Amazon ECR にプッシュされると暗号化署名を自動的に生成し、コンテナイメージの署名を簡素化します。詳細については、「マネージド署名」を参照してください。

Amazon ECR の開始方法

Amazon Elastic Container Service (Amazon ECS) または Amazon Elastic Kubernetes Service (Amazon EKS) を使用している場合、これらの 2 つのサービスの設定は Amazon ECR の設定と似ていることに注意してください。これは、Amazon ECR が両方のサービスの拡張機能であるためです。

Amazon ECR AWS Command Line Interface で を使用する場合は、最新の Amazon ECR 機能 AWS CLI をサポートする のバージョンを使用します。に Amazon ECR 機能のサポートが表示されない場合は AWS CLI、最新バージョンの にアップグレードします AWS CLI。の最新バージョンのインストールについては AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 の最新バージョンのインストールまたは更新 AWS CLI」を参照してください。

AWS CLI および Docker を使用してコンテナイメージをプライベート Amazon ECR リポジトリにプッシュする方法については、「Amazon ECR におけるライフサイクルを通じたイメージの移行」を参照してください。

Amazon ECR の料金表

Amazon ECR では、リポジトリに保存するデータ量、イメージのプッシュとプルからのデータ転送、イメージ署名やレプリケーションなど、オプトインしたイメージアクションに対して料金が発生します。詳細については、Amazon ECR の料金を参照してください。

Amazon ECR におけるライフサイクルを通じたイメージの移行

Amazon ECR を初めて使用する場合は、Docker CLI と で次の手順を使用してサンプルイメージ AWS CLI を作成し、デフォルトのレジストリで認証して、プライベートリポジトリを作成します。その後、プライベートリポジトリにイメージをプッシュし、プライベートリポジトリからイメージをプルします。サンプルイメージでの作業が完了したら、サンプルイメージとリポジトリを削除します。

AWS マネジメントコンソール の代わりに を使用するには AWS CLI、「」を参照してくださいthe section called “イメージを保存するためのリポジトリの作成”。

さまざまな AWS SDKs、IDE ツールキット、Windows PowerShell コマンドラインツールなど、AWS リソースの管理に使用できるその他のツールの詳細については、http://aws.amazon.com/tools/ を参照してください。

前提条件

最新の AWS CLI と Docker がインストールされておらず、使用できる状態になっていない場合は、次のステップを使用してこれらのツールの両方をインストールします。

のインストール AWS CLI

Amazon ECR AWS CLI で を使用するには、 AWS CLI 最新バージョンをインストールします。詳細については、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interfaceのインストール」を参照してください。

Docker をインストールする

Docker は、Ubuntu のような最新の Linux ディストリビューションから macOS や Windows まで、さまざまなオペレーティングシステムで使用できます。特定のオペレーティングシステムに Docker をインストールする方法の詳細については、Docker インストールガイド を参照してください。

Docker を使用するには、ローカルの開発システムは必要ありません。Amazon EC2 をすでに使用している場合は、Amazon Linux 2023 インスタンスを起動し、Docker をインストールして開始できます。

Docker をインストール済みの場合は、この手順をスキップして「ステップ 1: Docker イメージを作成する」に進んでください。

Amazon Linux 2023 AMI を使用して Amazon EC2 インスタンスに Docker をインストールするには

1. 最新の Amazon Linux 2023 AMI を使用してインスタンスを起動します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスの起動」を参照してください。
2. 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「Linux インスタンスへの接続」を参照してください。
3. インスタンスでインストールされているパッケージとパッケージキャッシュを更新します。

```
sudo yum update -y
```

4. 最新の Docker Community Edition パッケージをインストールします。

```
sudo yum install docker
```

5. Docker サービスを開始します。

```
sudo service docker start
```

6. ec2-user を docker グループに追加すると、sudo を使用せずに Docker コマンドを実行できます。

```
sudo usermod -a -G docker ec2-user
```

7. ログアウトし、再びログインして、新しい docker グループアクセス権限を取得します。これは、現在の SSH ターミナルウィンドウを閉じて、新しいウィンドウでインスタンスに再接続することで達成できます。新しい SSH セッションには適切な docker グループ権限があります。
8. ec2-user が sudo を使用せずに Docker コマンドを実行できることを確認します。

```
docker info
```

> **Note**
>
> 場合によっては、Docker デーモンにアクセスするための ec2-user に対するアクセス権限を提供するため、インスタンスを再起動する必要があります。次のエラーが表示された場合は、インスタンスを再起動してください。

```
Cannot connect to the Docker daemon. Is the docker daemon running on this
 host?
```

ステップ 1: Docker イメージを作成する

このステップでは、シンプルなウェブアプリケーションの Docker イメージを作成し、ローカルシステムまたは Amazon EC2 インスタンスでテストします。

シンプルなウェブアプリケーションの Docker イメージを作成するには

1. `Dockerfile` という名前のファイルを作成します。Dockerfile は、Docker イメージに使用する基本イメージと、そのイメージにインストールして実行するものを記述するマニフェストです。Dockerfile の詳細については、「Dockerfile リファレンス」を参照してください。

   ```
   touch Dockerfile
   ```

2. 前の手順で作成した `Dockerfile` を編集し、以下のコンテンツを追加します。

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest

   # Install dependencies
   RUN yum update -y && \
    yum install -y httpd

   # Install apache and write hello world message
   RUN echo 'Hello World!' > /var/www/html/index.html

   # Configure apache
   RUN echo 'mkdir -p /var/run/httpd' >> /root/run_apache.sh && \
    echo 'mkdir -p /var/lock/httpd' >> /root/run_apache.sh && \
    echo '/usr/sbin/httpd -D FOREGROUND' >> /root/run_apache.sh && \
    chmod 755 /root/run_apache.sh

   EXPOSE 80

   CMD /root/run_apache.sh
   ```

この Dockerfile は、Amazon ECR パブリックでホストされているパブリック Amazon Linux 2 イメージを使用します。RUN の手順により、パッケージキャッシュが更新され、ウェブ サーバー用のいくつかのソフトウェアがインストールされてから、「Hello World!」 のコンテンツがウェブサーバーのドキュメントルートに書き込みされます。EXPOSE の命令はコンテナ上のポート 80 を公開し、CMD の命令はウェブサーバーを起動します。

3. Dockerfile から Docker イメージを作成します。

> **Note**
>
> Docker の一部のバージョンでは、下に示す相対パスの代わりに、次のコマンドで Dockerfile への完全パスが必要になる場合があります。

```
docker build -t hello-world .
```

4. コンテナイメージを一覧表示します。

```
docker images --filter reference=hello-world
```

出力:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 194MB
```

5. 新しく構築されたイメージを実行します。-p 80:80 オプションは、コンテナ上の公開されたポート 80 をホストシステム上のポート 80 にマッピングします。docker run の詳細については、「Docker run reference」を参照してください。

```
docker run -t -i -p 80:80 hello-world
```

> **Note**
>
> Apache ウェブサーバーからの出力はターミナルウィンドウに表示されます。"Could not reliably determine the fully qualified domain name" メッセージは無視できます。

6. ブラウザーを開き、Docker を実行している、コンテナのホストサーバーを参照します。

   - EC2 インスタンスを使用している場合、これはサーバーの [Public DNS] 値であり、SSH でインスタンスに接続するときに使用するアドレスと同じです。インスタンスのセキュリティグループでポート 80 上の受信トラフィックを許可していることを確認します。
   - Docker をローカルに実行している場合は、ブラウザで http://localhost/ を参照します。
   - Windows または Mac コンピューターで docker-machine を使用している場合は、docker-machine ip コマンドを使用して Docker のホスト VirtualBox VM の IP アドレスを見つけ、*machine-name* を、使用中の Docker マシンの名前に置き換えます。

     ```
     docker-machine ip machine-name
     ```

   ウェブページに「Hello, World!」が確認できます。表示されます。

7. [Ctrl + C] キーを押して、Docker コンテナを停止します。

ステップ 2: リポジトリを作成する

これで Amazon ECR にプッシュするイメージが用意できたので、それを保持するレポジトリを作成する必要があります。この例では、`hello-repository` イメージを後でプッシュする、`hello-world:latest` と呼ばれるレポジトリを作成します。レポジトリを作成するには、次のコマンドを実行します。

```
aws ecr create-repository \
    --repository-name hello-repository \
    --region region
```

ステップ 3: デフォルトレジストリに対して認証する

をインストールして設定したら AWS CLI、デフォルトのレジストリに対して Docker CLI を認証します。これにより、docker コマンドは Amazon ECR を使用してイメージをプッシュおよびプルできます。 AWS CLI は、認証プロセスを簡素化するためのget-login-passwordコマンドを提供します。

get-login-password を使用して Amazon ECR レジストリに対して Docker を認証するには、aws ecr get-login-password コマンドを実行します。認証トークンを docker login コマンドに渡すとき、ユーザー名に AWS 値を使用し、認証先の Amazon ECR レジストリの URI を指定します。複数のレジストリに対して認証する場合は、レジストリごとにコマンドを繰り返す必要があります。

> **Important**
>
> エラーが発生した場合は、 AWS CLIの最新バージョンをインストールまたはアップグレードします。詳細については、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interfaceのインストール」を参照してください。

- get-login-password (AWS CLI)

```
aws ecr get-login-password --region region | docker login --username AWS --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

- Get-ECRLoginCommand (AWS Tools for Windows PowerShell)

```
(Get-ECRLoginCommand).Password | docker login --username AWS --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

ステップ 4: イメージを Amazon ECR にプッシュする

ここで、前のセクションで作成した Amazon ECR リポジトリにイメージをプッシュできます。次の前提条件が満たされた後に、docker CLI を使用してイメージをプッシュします。

- docker の最小バージョン 1.7 がインストールされている。
- Amazon ECR 認可トークンが docker login を使用して設定されている。
- Amazon ECR リポジトリが存在し、リポジトリにプッシュするアクセス権がユーザーにある。

これらの前提条件が満たされたら、アカウントのデフォルトレジストリで新しく作成されたレポジトリにイメージをプッシュできます。

イメージにタグを付けて Amazon ECR にプッシュするには

1. ローカルに保存したイメージをリストし、タグを付けてプッシュするイメージを識別します。

   ```
   docker images
   ```

   出力:

   ```
   REPOSITORY          TAG                 IMAGE ID            CREATED
    VIRTUAL SIZE
   hello-world         latest              e9ffedc8c286        4 minutes ago
    241MB
   ```

2. リポジトリにプッシュするイメージにタグを付けます。

   ```
   docker tag hello-world:latest aws_account_id.dkr.ecr.region.amazonaws.com/hello-
   repository
   ```

3. イメージをプッシュします。

   ```
   docker push aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository:latest
   ```

   出力:

   ```
   The push refers to a repository [aws_account_id.dkr.ecr.region.amazonaws.com/hello-
   repository] (len: 1)
   e9ae3c220b23: Pushed
   a6785352b25c: Pushed
   0998bf8fb9e9: Pushed
   0a85502c06c9: Pushed
   latest: digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636EXAMPLE
    size: 6774
   ```

ステップ 5: Amazon ECR からイメージをプルする

イメージを Amazon ECR リポジトリにプッシュした後、そのイメージを他の場所からプルできます。次の前提条件が満たされた後に、docker CLI を使用してイメージをプルします。

- docker の最小バージョン 1.7 がインストールされている。
- Amazon ECR 認可トークンが docker login を使用して設定されている。
- Amazon ECR リポジトリが存在し、リポジトリからプルするアクセス許可がユーザーにある。

これらの前提条件が満たされたら、イメージをプルできます。Amazon ECR からサンプルイメージをプルするには、次のコマンドを実行します。

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository:latest
```

出力:

```
latest: Pulling from hello-repository
0a85502c06c9: Pull complete
0998bf8fb9e9: Pull complete
a6785352b25c: Pull complete
e9ae3c220b23: Pull complete
Digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636EXAMPLE
Status: Downloaded newer image for aws_account_id.dkr.region.amazonaws.com/hello-
repository:latest
```

ステップ 6: イメージを削除する

いずれかのリポジトリでイメージが不要になった場合は、そのイメージを削除できます。イメージを削除するには、そのイメージが配置されているリポジトリと、そのイメージの `imageTag` または `imageDigest` の値を指定します。次の例では、`hello-repository` リポジトリ内でイメージタグ `latest` が付いているイメージを削除します。リポジトリからサンプルイメージを削除するには、次のコマンドを実行します。

```
aws ecr batch-delete-image \
      --repository-name hello-repository \
      --image-ids imageTag=latest \
      --region region
```

ステップ 7: リポジトリを削除する

イメージのリポジトリ全体が不要になった場合は、リポジトリを削除できます。次の例では、`--force` フラグを使用して、イメージが含まれるリポジトリを削除します。イメージ (およびその中に含まれるすべてのイメージ) を含むリポジトリを削除するには、次のコマンドを実行します。

```
aws ecr delete-repository \
      --repository-name hello-repository \
      --force \
      --region region
```

Amazon ECR のパフォーマンスを最適化する

設定および戦略に関する次の推奨事項を使用して、Amazon ECR を使用する際のパフォーマンスを最適化できます。

レイヤーの同時アップロードを利用するには、Docker 1.10 以降を使用する

Docker イメージは、イメージの中間ビルドステージであるレイヤーで構成されます。Dockerfile の各行により、新しいレイヤーが作成されます。Docker 1.10 以降を使用する場合、Docker はデフォルトで可能な限り多くのレイヤーを Amazon ECR への同時アップロードとしてプッシュし、それによりアップロード時間が高速になります。

小さなベースイメージを使用する

Docker Hub を通じて利用できるデフォルトイメージには、アプリケーションが要求しない多くの依存関係が含まれている場合があります。Docker コミュニティで他のユーザーによって作成、管理される、より小さいイメージを使用するか、Docker の最小スクラッチイメージを使用して独自のベースイメージを構築することを検討します。詳細については、Docker のドキュメントの「ベースイメージの作成」を参照してください。

Dockerfile で、最も変更の少ない依存関係を前に配置する

Docker はレイヤーをキャッシュし、それによりビルド時間を高速化します。最後のビルド以降にレイヤーで何も変更されていない場合、Docker はレイヤーを再構築する代わりにキャッシュしたバージョンを使用します。ただし、各レイヤーは、それ以前のレイヤーに依存しています。レイヤーが変更された場合、Docker はそのレイヤーだけでなく、そのレイヤーの後のレイヤーも同様に再コンパイルします。

Docker ファイルの再構築と、レイヤーの再アップロードに必要な時間を最小化するため、最も変更を行わない依存関係を Dockerfile で前に配置します。頻繁に変更を行う依存関係 (アプリケーションのソースコードなど) はスタックで後に配置します。

コマンドをチェーン処理して不要なファイルの保存を避ける

レイヤーで作成された中間ファイルは、それ以降のレイヤーで削除された場合でも、そのレイヤーの一部として残ります。次の例を考えます。

```
WORKDIR /tmp
RUN wget http://example.com/software.tar.gz
RUN wget tar -xvf software.tar.gz
RUN mv software/binary /opt/bin/myapp
```

```
RUN rm software.tar.gz
```

この例では、最初と 2 番目の RUN コマンドによって作成されたレイヤーには、元の .tar.gz ファイルと解凍されていないそのすべてのコンテンツが含まれます。ただし、.tar.gz ファイルは 4 番目の RUN コマンドによって削除されます。これらのコマンドを 1 つの RUN ステートメントにチェーン処理して、不要なファイルが最終的な Docker イメージの一部とはならないようにできます。

```
WORKDIR /tmp
RUN wget http://example.com/software.tar.gz &&\
    wget tar -xvf software.tar.gz &&\
    mv software/binary /opt/bin/myapp &&\
    rm software.tar.gz
```

最も近いリージョンのエンドポイントを使用する

アプリケーションが実行されている場所に最も近いリージョンのエンドポイントを使用することにより、Amazon ECR からのイメージのプルのレイテンシーを減らすことができます。アプリケーションが Amazon EC2 インスタンスで実行されている場合、次のシェルコードを使用して、インスタンスのアベイラビリティーゾーンからリージョンを取得できます。

```
REGION=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone
 |\
     sed -n 's/\(\d*\)[a-zA-Z]*$/\1/p')
```

リージョンは AWS CLI パラメーターを使用して --region コマンドに渡すか、aws configure コマンドを使用してプロファイルのデフォルトリージョンとして設定できます。また、AWS SDK を使用して呼び出しを行うときに、リージョンを設定することもできます。詳細については、ご使用の特定のプログラミング言語の SDK のドキュメントを参照してください。

Amazon ECR レジストリにリクエストを実行する

IPv4 のみのエンドポイントまたはデュアルスタック (IPv4 と IPv6) エンドポイントを使用して。Amazon ECR プライベートレジストリの OCI イメージ、Docker イメージ、OCI 互換アーティファクトをプッシュ、プル、削除、表示、管理できます。IPv4 ネットワークからのリクエストには、デュアルスタックエンドポイントまたは IPv4 エンドポイントを使用できます。IPv6 ネットワークからリクエストを行うには、デュアルスタックのエンドポイントを使用します。IPv4 エンドポイントとデュアルスタックエンドポイントを使用して Amazon ECR パブリックレジストリにリクエストを行う方法の詳細については、「Making requests to Amazon ECR Public registries」を参照してください。IPv6 を使用して Amazon ECR にアクセスする場合、追加料金はかかりません。詳細については、「Amazon Elastic Container Registry の料金」を参照してください。

Amazon ECR エンドポイントは、IPv4 のみのエンドポイントまたはデュアルスタックエンドポイントのサポート以上の属性によって指定されます。これらの属性には以下が含まれます。

- リージョン – 各エンドポイントはリージョンに固有です。
- タイプ – エンドポイントの選択は、 AWS SDK と OCI 互換の Docker コマンドラインインターフェイスのどちらを使用しているかによって異なります。
- セキュリティ – 一部のリージョンでは、Amazon ECR は FIPS 準拠のエンドポイントを提供します。FIPS 互換の Amazon ECR エンドポイントの詳細については、「Federal Information Processing Standard (FIPS) 140-3」を参照してください。

CLI および SDK からの Amazon ECR API コールを処理する IPv4、デュアルスタック、Docker、および OCI AWS クライアントでサポートされるサービスエンドポイントの詳細については、「サービスエンドポイント」を参照してください。 AWS SDKs

IPv6 を使用したリクエストの実行の開始方法

IPv6 を使用して Amazon ECR レジストリにリクエストを実行するには、デュアルスタックのエンドポイントを使用する必要があります。IPv6 経由で Amazon ECR レジストリにアクセスする前に、次の要件を確認してください。

- クライアントとネットワークは IPv6 をサポートする必要があります。
- Amazon ECR は、IPv6 経由で次のリクエストタイプをサポートします。
  - OCI および Docker クライアントリクエスト:

`<registry-id>.dkr-ecr.<aws-region>.on.aws`

- AWS API リクエスト:

  `ecr.<aws-region>.api.aws`

- IPv6 アドレス範囲を含めるには、ソース IP アドレスフィルタリングを使用する AWS Identity and Access Management (IAM) またはレジストリポリシーを更新する必要があります。詳細については、「IAM ポリシーでの IPv6 アドレスの使用」を参照してください。
- IPv6 を使用すると、サーバーアクセスログに IPv6 形式の `Remote IP` アドレスが表示されます。これらの IPv6 形式の IP アドレスを解析するには、既存のツール、スクリプト、ソフトウェアを更新します。

> **Note**
>
> ログファイルで IPv6 アドレスに関連する問題を見つけた場合、AWS サポート にお問い合わせください。

IP アドレス互換性のテスト

Linux/Unix または Mac OS X を使用している場合、次の例に示すように `curl` コマンドを使用して IPv6 でデュアルスタックのエンドポイントにアクセスできるかどうかテストできます。

Example

```
curl --verbose https://ecr.us-west-2.api.aws
```

次の例のような情報を取得できます。IPv6 で接続している場合、接続されている IP アドレスは IPv6 アドレスです。

```
* About to connect() to ecr.us-west-2.api.aws port 443 (#0)
* Trying IPv6 address... connected
* Connected to ecr.us-west-2.api.aws (IPv6 address) port 443 (#0)
> Host: ecr.us-west-2.api.aws
* Request completely sent off
```

Microsoft Windows 7 または Windows 10 を使用している場合、次の例に示すように ping コマンドを使用して IPv4 または IPv6 でデュアルスタックのエンドポイントにアクセスできるかどうかテストできます。

```
ping ecr.us-west-2.api.aws
```

デュアルスタックのエンドポイントを使用して IPv6 経由でリクエストを実行する

Amazon ECR API コールは、デュアルスタックエンドポイントを使用して IPv6 経由で行うことができます。Amazon ECR API オペレーションの機能とパフォーマンスは、IPv4 と IPv6 のどちらを使用する場合でも一貫しています。

AWS Command Line Interface (AWS CLI) と AWS SDKs を使用する場合、IPv6 を有効にするには、パラメータまたはフラグを使用してデュアルスタックエンドポイントに切り替えるか、設定ファイルでデュアルスタックエンドポイントを直接指定してデフォルトの Amazon ECR エンドポイントを上書きします。デフォルトプロファイルで use_dualstack_endpoint を true に設定するコマンドを使用して設定を変更することもできます。use_dualstack_endpoint の詳細については、「Dual-stack and FIPS endpoints」を参照してください。

Exampleコマンドを使用して設定を変更する

```
aws configure set default.ecr.use_dualstack_endpoint true
```

Exampleを使用した IPv6 経由のリクエストの実行 AWS CLI

```
aws ecr describe-repositories --region us-west-2 --endpoint-url https://
ecr.us-west-2.api.aws
```

Docker CLI から Amazon ECR エンドポイントを使用する

Amazon ECR リポジトリにサインインしてイメージにタグを付けると、Amazon ECR レジストリに対して OCI イメージと Docker イメージをプッシュおよびプルできます。次の例は、両方のデュアルスタックエンドポイントを使用した docker push コマンドと docker pull コマンドを示しています。

Example IPv4 エンドポイントを使用して Docker イメージをプッシュする

```
docker push <registry-id>.dkr.ecr.us-west-1.amazonaws.com/my-repository:tag
```

Exampleデュアルスタックエンドポイントを使用して Docker イメージをプッシュする

```
docker push <registry-id>.dkr-ecr.us-west-1.on.aws/my-repository:tag
```

Example IPv4 エンドポイントを使用して Docker イメージをプルする

```
docker pull <registry-id>.dkr.ecr.us-west-1.amazonaws.com/my-repository:tag
```

Exampleデュアルスタックエンドポイントを使用して Docker イメージをプルする

```
docker pull <registry-id>.dkr-ecr.us-west-1.on.aws/my-repository:tag
```

IAM ポリシーでの IPv6 アドレスの使用

IPv6 を使用してレジストリにアクセスする前に、IP アドレスフィルタリングを使用する IAM ユーザーと Amazon ECR レジストリポリシーに IPv6 アドレス範囲が含まれていることを確認してください。IPv6 アドレスを処理するように IP アドレスのフィルタリングポリシーが更新されていない場合、クライアントが IPv6 の使用を開始する際、クライアントによるバケットへのアクセスの取得または損失が誤って発生することがあります。IAM でのアクセス許可管理の詳細については、「Amazon Elastic Container Registry の Identity and Access Management」を参照してください。

IP アドレスをフィルタリングする IAM ポリシーは、IP アドレス条件演算子を使用します。次のレジストリポリシーの例は、IP アドレス条件演算子を使用して、許可された IPv4 アドレスの `54.240.143.*` 範囲を識別する方法を示します。この範囲外のすべての IP アドレスはレジストリ (`exampleregistry`) へのアクセスを拒否されます。すべての IPv6 アドレスは許可範囲外であるため、このポリシーは IPv6 アドレスの `exampleregistry` へのアクセスをブロックします。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "ecr:*",
      "Resource": "arn:aws:ecr:*:*:repository/exampleregistry/*",
      "Condition": {
         "IpAddress": {"aws:SourceIp": "54.240.143.0/24"}
```

```
      }
    }
  ]
}
```

IPv4 (54.240.143.0/24) と IPv6 (2001:DB8:1234:5678::/64) の両方のアドレス範囲を許可するには、次の例に示すようにレジストリポリシーの Condition 要素を変更します。この Condition ブロック形式を使用して、IAM ユーザーポリシーとレジストリポリシーの両方を更新できます。

```
"Condition": {
  "IpAddress": {
     "aws:SourceIp": [
       "54.240.143.0/24",
        "2001:DB8:1234:5678::/64"
      ]
   }
 }
```

Important

IPv6 を使用する前に、IP アドレスのフィルタリングを使用するすべての関連 IAM ユーザーとレジストリポリシーを更新する必要があります。レジストリポリシーで IP アドレスのフィルタリングを使用することは推奨されません。

IAM ユーザーポリシーは、https://console.aws.amazon.com/iam/ の IAM コンソールを使用して確認できます。IAM の詳細については、IAM ユーザーガイドを参照してください。

Amazon ECR プライベートレジストリ

Amazon ECR プライベートレジストリは、可用性の高いスケーラブルなアーキテクチャでコンテナイメージをホストします。プライベートレジストリを使用して、Docker イメージ、Open Container Initiative (OCI) イメージ、アーティファクトで構成されるプライベートイメージリポジトリを管理できます。各 AWS アカウントには、デフォルトのプライベート Amazon ECR レジストリが提供されます。Amazon ECR パブリックレジストリの詳細については、Amazon Elastic Container Registry Public ユーザーガイドの「Public registries」を参照してください。

プライベートレジストリの概念

- デフォルトのプライベートレジストリの URL は `https://aws_account_id.dkr.ecr.region.amazonaws.com` です。
- デフォルトでは、アカウントにはプライベートリポジトリ内のリポジトリへの読み取りおよび書き込み許可があります。ただし、ユーザーには、Amazon ECR API への呼び出しと、プライベートリポジトリに対するイメージのプッシュまたはプルを行うためのアクセス許可が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するための複数の 管理ポリシーが用意されています。詳細については、「Amazon Elastic Container Registry のアイデンティティベースのポリシーの例」を参照してください。
- プライベートレジストリに対して Docker クライアントを認証し、docker push コマンドと docker pull コマンドを使用してそのレジストリ内のリポジトリとの間でイメージをプッシュおよびプルできるようにする必要があります。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。
- プライベートリポジトリは、 ユーザーアクセスポリシーとリポジトリポリシーによって制御できます。リポジトリポリシーの詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。
- プライベートレジストリのリポジトリは、プライベートレジストリのレプリケーションを設定することで、プライベートレジストリ内の AWS リージョン、および個別のアカウントにわたってレプリケートできます。詳細については、「Amazon ECR でのプライベートイメージレプリケーション」を参照してください。

Amazon ECR でのプライベートレジストリ認証

AWS マネジメントコンソール、 AWS CLI、または AWS SDKs を使用して、プライベートリポジトリを作成および管理できます。また、これらの方法を使用して、イメージの一覧表示や削除などのい

くつかのアクションをイメージで実行できます。これらのクライアントは、標準の AWS 認証方法を使用します。Amazon ECR API を使用してイメージをプッシュおよびプルできますが、Docker CLI または言語固有の Docker ライブラリの使用をお勧めします。

Docker CLI は、ネイティブの IAM 認証方法をサポートしていません。Amazon ECR が、Docker のプッシュ要求とプル要求を認証および認可できるようにするには、追加の手順を実行する必要があります。

以下のセクションで説明するレジストリ認証方法を使用できます。

Amazon ECR 認証情報ヘルパーを使用する

Amazon ECR には Docker 認証情報ヘルパーが用意されているので、Amazon ECR に対してイメージをプッシュおよびプルするときに、Docker 認証情報をより簡単に保存および使用できます。インストールおよび設定手順については、「Amazon ECR Docker 認証情報ヘルパー」を参照してください。

> **Note**
>
> Amazon ECR Docker 認証情報ヘルパーは、現在、多要素認証 (MFA) をサポートしていません。

認可トークンを使用する

認可トークンの許可スコープは、認証トークンの取得に使用される IAM プリンシパルの許可スコープと一致します。認証トークンは、IAM プリンシパルからアクセス可能で 12 時間有効な Amazon ECR レジストリにアクセスするために使用されます。認可トークンを取得するには、GetAuthorizationToken API オペレーションを使用して、ユーザー名 `AWS` とエンコードされたパスワードを含む base64 エンコード認可トークンを取得します。コマンドは AWS CLI `get-login-password`、認証トークンを取得してデコードすることでこれを簡素化し、認証するdocker loginコマンドにパイプできます。

get-login を使用して Amazon ECR プライベートレジストリに対して Docker を認証するには

- get-login-password を使用して Amazon ECR レジストリに対して Docker を認証するには、aws ecr get-login-password コマンドを実行します。認証トークンを docker login コマンドに渡すとき、ユーザー名に `AWS` 値を使用し、認証先の Amazon ECR レジストリの URI を指定します。

複数のレジストリに対して認証する場合は、レジストリごとにコマンドを繰り返す必要があります。

> **Important**
>
> エラーが発生した場合は、 AWS CLIの最新バージョンをインストールまたはアップグレードします。詳細については、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interfaceのインストール」を参照してください。

- get-login-password (AWS CLI)

```
aws ecr get-login-password --region region | docker login --username AWS --
password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

- Get-ECRLoginCommand (AWS Tools for Windows PowerShell)

```
(Get-ECRLoginCommand).Password | docker login --username AWS --password-
stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

HTTP API 認証を使用する

Amazon ECR は Docker Registry HTTP API をサポートします。ただし、Amazon ECR はプライベートレジストリであるため、すべての HTTP リクエストで認可トークンを提供する必要があります。の -Hオプションを使用して HTTP 認可ヘッダーを追加curlし、 get-authorization-token AWS CLI コマンドによって提供される認可トークンを渡すことができます。

Amazon ECR HTTP API を使用して認証するには

1. AWS CLI を使用して認可トークンを取得し、そのトークンを環境変数に設定します。

```
TOKEN=$(aws ecr get-authorization-token --output text --query
 'authorizationData[].authorizationToken')
```

2. API に対して認証するには、$TOKEN 変数を curl の -H オプションに渡します。たとえば、次のコマンドでは、Amazon ECR レポジトリでイメージタグを一覧表示します。詳細については、Docker レジストリ HTTP API リファレンスドキュメントを参照してください。

```
curl -i -H "Authorization: Basic $TOKEN"
 https://aws_account_id.dkr.ecr.region.amazonaws.com/v2/amazonlinux/tags/list
```

出力は次のとおりです。

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Date: Thu, 04 Jan 2018 16:06:59 GMT
Docker-Distribution-Api-Version: registry/2.0
Content-Length: 50
Connection: keep-alive

{"name":"amazonlinux","tags":["2017.09","latest"]}
```

Amazon ECR でのプライベートレジストリ設定

Amazon ECR はプライベートレジストリ設定を使用してレジストリレベルで機能を構成します。プライベートレジストリ設定は、リージョンごとに個別に構成されます。プライベートレジストリ設定を使用して、次の機能を設定できます。

- [レジストリのアクセス許可] — レジストリのアクセス許可ポリシーは、レプリケーションアクセス許可とプルスルーキャッシュアクセス許可を制御します。詳細については、「Amazon ECR でのプライベートレジストリのアクセス許可」を参照してください。
- [プルスルーキャッシュルール] — プルスルーキャッシュルールを使用して、Amazon ECR プライベートレジストリのアップストリームレジストリからイメージをキャッシュします。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリの同期」を参照してください。
- [レプリケーション設定] — レプリケーション設定は、リポジトリのコピー方法 (AWS リージョンの間または AWS アカウントの間) を制御するために使用されます。詳細については、「Amazon ECR でのプライベートイメージレプリケーション」を参照してください。
- [リポジトリ作成テンプレート] — リポジトリ作成テンプレートは、Amazon ECR がユーザーに代わって新しいリポジトリを作成したときに適用される標準設定を定義するために使用されます。たとえば、プルスルーキャッシュアクション、プッシュ時作成、レプリケーションによって作成されたリポジトリなどです。詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。

- [スキャン設定] — デフォルトでは、レジストリでベーシックスキャンが有効です。オペレーティングシステムとプログラミング言語パッケージの脆弱性の両方をスキャンする自動連続スキャンモードを提供する拡張スキャンを有効にできます。詳細については、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。
- プルタイム更新の除外 – プルタイム更新の除外を設定して、プルされた特定のイメージの最後のプルタイムが更新されないようにできます。これは、プル時間がライフサイクルポリシーの決定に影響を与えたくない、テストまたは CI/CD 目的で使用されるイメージに役立ちます。詳細については、「プルタイム更新の除外」を参照してください。
- BLOB マウント設定 – BLOB マウント設定は、レジストリ内のリポジトリが重複レイヤーを保存するのではなく、共通レイヤーを共有するかどうかを制御するために使用されます。詳細については、「Amazon ECR での Blob マウント」を参照してください。

Amazon ECR での Blob マウント

Amazon ECR は、BLOB マウントと呼ばれる機能をサポートし、レジストリ内のリポジトリ間で共通のイメージレイヤーを共有します。有効にすると、1 つのレジストリ内のリポジトリは、重複したコピーを保存する代わりに、同じレジストリ内の他のリポジトリのレイヤーを参照できます。

レジストリ BLOB マウントが有効になっている場合、Amazon ECR は、マウントパラメータが含まれている場合、プッシュオペレーション中にレジストリ内の既存のレイヤーをチェックします。同じレジストリ内の別のリポジトリにレイヤーがすでに存在する場合、Amazon ECR は重複をアップロードする代わりに既存のレイヤーをマウントします。

> **Note**
>
> OCI クライアントは、BLOB が別のリポジトリに既に存在する可能性があることを検出すると、マウントパラメータを自動的に含めます。Amazon ECR は、これらのパラメータがクライアントの POST リクエストに存在する場合にのみマウントを試みます。

Blob マウントの概念

- BLOB マウントは、同じレジストリ (同じアカウントとリージョン) 内でのみ機能します。
- リポジトリは、同じ暗号化タイプとキーを使用する必要があります。
- Blob マウントは、プルスルーキャッシュで作成されたイメージではサポートされていません。

- BLOB マウントを無効にすると、BLOB マウントが設定された既存のイメージは引き続き機能し、レイヤーはマウントされたままになります。

Blob マウント設定

AWS マネジメントコンソール または を使用して AWS CLI 、レジストリの BLOB マウントを設定できます。

> **Note**
> ユーザーには、リポジトリからレイヤーをマウントするための `ecr:GetDownloadUrlForLayer` IAM アクセス許可が必要です。

AWS マネジメントコンソール

を使用してレジストリの BLOB マウント設定を更新するには、次の手順に従います AWS マネジメントコンソール。

プライベートレジストリの BLOB マウント設定を有効にする

1. https://console.aws.amazon.com/ecr/private-registry/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーからリージョンを選択します。
3. ナビゲーションペインで、プライベートレジストリ、特徴と設定を選択し、BLOB マウントを選択します。
4. Blob マウントページで、有効化を選択します。

   BLOB マウント設定が有効に更新されたことを示すバナーが表示されます。

AWS CLI

次のコマンドを使用して、 を使用してレジストリの BLOB マウント設定を更新します AWS CLI。

- ```
  aws ecr put-account-setting --name BLOB_MOUNTING --value ENABLED
  ```
  ```

Amazon ECR でのプライベートレジストリのアクセス許可

Amazon ECR はレジストリポリシーを使用して、プライベートレジストリレベルの AWS プリンシパルに許可を付与します。

スコープは、レジストリポリシーのバージョンを選択することによって設定されます。レジストリポリシーの範囲が異なるバージョン 1 (V1) とバージョン 2 (V2) という 2 つのバージョンがあります。V2 は、すべての ECR アクセス許可を含む拡張レジストリポリシースコープです。API アクションの完全なリストについては、「Amazon ECR API ガイド」を参照してください。V2 バージョンはデフォルトのレジストリポリシースコープです。レジストリポリシースコープの表示または設定の詳細については、「拡張レジストリポリシースコープに切り替える」を参照してください。Amazon ECR プライベートレジストリの一般的な設定については、「Amazon ECR でのプライベートレジストリ設定」を参照してください。

バージョンの詳細は次のとおりです。

- V1 – バージョン 1 の場合、Amazon ECR では、プライベートレジストリレベルでのみ以下の許可が適用されます。
  - `ecr:ReplicateImage` – ソースレジストリと呼ばれる別のアカウントに、そのイメージをレジストリにレプリケートする許可を付与します。これは、クロスアカウントレプリケーションのみに使用されます。
  - `ecr:BatchImportUpstreamImage` – 外部イメージを取得し、プライベートレジストリにインポートするアクセス許可を付与します。
  - `ecr:CreateRepository` – プライベートレジストリにリポジトリを作成するアクセス許可を付与します。レプリケートまたはキャッシュされたイメージを保存するリポジトリがプライベートレジストリにまだ存在しない場合は、この許可が必要となります。
- V2 – バージョン 2 の場合、Amazon ECR はポリシー内のすべての ECR アクションを許可し、すべての ECR リクエストにレジストリポリシーを適用します。

コンソールまたは CLI を使用して、レジストリポリシースコープを表示または変更できます。

> **Note**
>
> プライベートレジストリポリシーに `ecr:*` アクションを追加することが可能である場合、ワイルドカードを使用するのではなく、使用している機能に基づいて必要な特定のアクションのみを追加することがベストプラクティスと考えられます。

トピック

- Amazon ECR のプライベートレジストリのポリシーの例
- 拡張レジストリポリシースコープに切り替える
- Amazon ECR でのクロスアカウントレプリケーション用のレジストリアクセス許可の付与
- Amazon ECR でのプルスルーキャッシュ用のレジストリアクセス許可の付与

Amazon ECR のプライベートレジストリのポリシーの例

以下の例では、Amazon ECR レジストリに対してユーザーが所有するアクセス許可を制御するために使用できるレジストリ許可ポリシーステートメントを示しています。

> **Note**
>
> それぞれの例では、レジストリポリシーから `ecr:CreateRepository` アクションが削除されていても、レプリケーションは引き続き発生する場合があります。ただし、レプリケーションを成功させるには、アカウント内に同じ名前のリポジトリを作成する必要があります。

例: ソースアカウントのすべての IAM プリンシパルにすべてのリポジトリのレプリケートを許可する

次のレジストリアクセス許可ポリシーは、ソースアカウントのすべての IAM プリンシパル (ユーザーとロール) がすべてのリポジトリをレプリケートすることを許可します。

次の点に注意してください。

- 重要: ポリシーで AWS アカウント ID をプリンシパルとして指定すると、ルートユーザーだけでなく、そのアカウント内のすべての IAM ユーザーとロールにアクセス許可が付与されます。その結果、アカウント全体で広範なアクセスが可能になります。
- セキュリティ上の考慮事項: アカウントレベルのアクセス許可では、指定されたアカウント内のすべての IAM エンティティへのアクセスが許可されます。アクセスの制限を厳しくするには、個々の IAM ユーザー、ロールを指定するか、条件ステートメントを使用してアクセスをさらに制限します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::111122223333:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:444455556666:repository/*"
            ]
        }
    ]
}
```

例: 複数のアカウントからの IAM プリンシパルを許可する

次のレジストリアクセス許可ポリシーには、2 つのステートメントがあります。各ステートメントでは、ソースアカウントのすべての IAM プリンシパル (ユーザーとロール) がすべてのリポジトリをレプリケートできます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount1",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::111122223333:root"
            },
            "Action":[
```

```
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:123456789012:repository/*"
            ]
        },
        {
            "Sid":"ReplicationAccessCrossAccount2",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::444455556666:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:123456789012:repository/*"
            ]
        }
    ]
}
```

例: ソースアカウントのすべての IAM プリンシパルにプレフィックス **prod-** のあるすべてのリポジトリのレプリケートを許可する

次のレジストリアクセス許可ポリシーは、ソースアカウントのすべての IAM プリンシパル (ユーザーとロール) が `prod-` で始まるすべてのリポジトリをレプリケートすることを許可します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::111122223333:root"
            },
```

```
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:444455556666:repository/prod-*"
            ]
        }
    ]
}
```

拡張レジストリポリシースコープに切り替える

> **Important**
>
> 新規ユーザーの場合、レジストリは作成時に、 V2 レジストリポリシーを使用するように自動的に設定されます。ユーザーが実行するアクションはありません。Amazon ECR では、以前のレジストリポリシー に戻すことはお勧めしませんV1。

コンソールまたは CLI を使用して、レジストリポリシースコープを表示または変更できます。

AWS マネジメントコンソール

次のステップに従ってアカウント設定を表示します。レジストリポリシースコープを表示または更新するには、このページの CLI の手順を参照してください。

プライベートレジストリに対して拡張レジストリポリシーを有効にする

1. https://console.aws.amazon.com/ecr/private-registry/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーからリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ]、[Feature & Settings]、[アクセス許可] の順に選択します。
4. アクセス [許可] ページで、[レジストリポリシー] のポリシー JSON を表示します。V1 ポリシーがある場合、バナーに V2 に更新する手順が表示されます。[有効化] を選択します。

   レジストリポリシースコープが V2 に更新されたことを示すバナーが表示されます。

5. CLI を使用してアクセス許可を設定することもできます。詳細については、「Amazon ECR でのプライベートレジストリ設定」を参照してください。

> **Note**
>
> レジストリポリシースコープを表示または更新するには、このページの CLI の手順を参照してください。

AWS CLI

Amazon ECR は V2 レジストリポリシーを生成します。レジストリポリシーの範囲を表示または更新するには、次のステップを使用します。コンソールでレジストリポリシースコープを表示または変更することはできません

- 現在使用しているレジストリポリシーを取得するには:

```
aws ecr get-account-setting --name REGISTRY_POLICY_SCOPE
```

パラメータ名は必須フィールドです。名前を指定しない場合、次のエラーが表示されます。

```
 aws: error: the following arguments are required: --name
```

レジストリポリシーコマンドの出力を表示します。次の出力例では、レジストリポリシーのバージョンは V1 です。

```
{
    "name": "REGISTRY_POLICY_SCOPE",
    "value": "V1"
}
```

レジストリポリシーのバージョンを V1 から V2 に変更できます。V1 は推奨されるレジストリポリシースコープではありません。

```
aws ecr put-account-setting --name REGISTRY_POLICY_SCOPE --value value
```

例えば、次のコマンドを使用して V2 に更新します。

```
aws ecr put-account-setting --name REGISTRY_POLICY_SCOPE --value V2
```

レジストリポリシーコマンドの出力を表示します。次の出力例では、レジストリポリシーのバージョンが V2 に更新されました。

```
{
    "name": "REGISTRY_POLICY_SCOPE",
    "value": "V2"
}
```

Amazon ECR でのクロスアカウントレプリケーション用のレジストリアクセス許可の付与

クロスアカウントポリシータイプは、プリン AWS シパルにアクセス許可を付与するために使用されます。これにより、ソースレジストリからレジストリへのリポジトリのレプリケーションが可能になります。デフォルトでは、ユーザーには、独自のレジストリ内でクロスリージョンレプリケーションを設定する許可があります。レジストリにコンテンツをレプリケートする許可を別のアカウントに付与する場合、必要な操作は、レジストリポリシーを構成することだけです。

レジストリポリシーは、`ecr:ReplicateImage` API アクションに許可を付与する必要があります。この API は、複数のリージョンまたはアカウント間でイメージをレプリケートできる内部 Amazon ECR API です。 `ecr:CreateRepository` の許可を付与することもできます。この場合、レポジトリが存在しない場合、Amazon ECR がレジストリ内にリポジトリを作成できるようになります。`ecr:CreateRepository` の許可が指定されていない場合、ソースリポジトリと同じ名前のリポジトリを手動でレジストリに作成する必要があります。どちらも行われていない場合、レプリケーションは失敗します。失敗した `CreateRepository` アクション または `ReplicateImage` API アクションが CloudTrail に表示されます。

レプリケーションの許可ポリシーを構成するには (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、レジストリポリシーを設定するリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ]、[Feature & Settings]、[アクセス許可] の順に選択します。
4. [Registry permissions] (レジストリー許可) ページで [Generate statement] (ステートメントを生成) を選択します。

5. ポリシージェネレータを使用してポリシーステートメントメントを定義するには、次の手順を実行します。

   a. [ポリシータイプ] で、[レプリケーション - クロスアカウント] を選択します。

   b. [ステートメント ID] に一意のステートメント ID を入力します。このフィールドは、レジストリポリシーの `Sid` として使用されます。

   c. [Accounts] (アカウント) に、許可を付与する各アカウントのアカウント ID を入力します。複数のアカウント ID を指定する場合は、カンマで区切ります。

6. [保存] を選択します。

レプリケーションの許可ポリシーを構成するには (AWS CLI)

1. `registry_policy.json` という名前のファイルを作成し、レジストリポリシーの情報を入力します。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::111122223333:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:444455556666:repository/*"
            ]
        }
    ]
}
```

2. ポリシーファイルを使用してレジストリポリシーを作成します。

```
aws ecr put-registry-policy \
      --policy-text file://registry_policy.json \
      --region us-west-2
```

3. レジストリのポリシーを取得して確認します。

```
aws ecr get-registry-policy \
      --region us-west-2
```

Amazon ECR でのプルスルーキャッシュ用のレジストリアクセス許可の付与

Amazon ECR プライベートレジストリ許可は、プルスルーキャッシュを使用する個々の IAM エンティティのアクセス許可のスコープ設定に使用できます。レジストリ許可ポリシーによって付与されるアクセス許可よりも多くのアクセス許可が IAM ポリシーによって IAM エンティティに付与される場合、IAM ポリシーが優先されます。

プライベートレジストリの許可ポリシーを作成するには (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ許可ステートメントを設定するリージョンを選択します。
3. ナビゲーションペインで、プライベートレジストリを選択し、機能と設定を選択し、アクセス許可を選択します。
4. [Registry permissions] (レジストリー許可) ページで [Generate statement] (ステートメントを生成) を選択します。
5. 作成するプルスルーキャッシュ許可ポリシーステートメントごとに、次の操作を行います。
   a. [ポリシータイプ] で、[プルスルーキャッシュポリシー] を選択します。
   b. [ステートメント ID] で、プルスルーキャッシュステートメントポリシーの名前を指定します。
   c. [IAM entities] (IAM エンティティ) で、ポリシーに含めるユーザー、グループ、またはロールを指定します。
   d. [キャッシュ名前空間] で、ポリシーを関連付けるプルスルーキャッシュルールを選択します。

e. [リポジトリ名] で、ルールを適用するリポジトリベース名を指定します。たとえば、Amazon ECR パブリックで Amazon Linux リポジトリを指定する場合、リポジトリ名は `amazonlinux` になります。

Amazon ECR プライベートリポジトリ

Amazon ECR プライベートリポジトリには、Docker イメージ、Open Container Initiative (OCI) イメージ、および OCI 互換アーティファクトが含まれます。Amazon ECR API オペレーションまたは Amazon ECR コンソールの [リポジトリ] セクションを使用して、イメージリポジトリの作成、監視、削除、およびそれらにアクセスできるユーザーを制御するアクセス許可の設定を行うことができます。また、Amazon ECR と Docker CLI の統合によって、開発環境とリポジトリの間でイメージをプッシュおよびプルすることもできます。

トピック

- プライベートリポジトリの概念
- イメージを保存するための Amazon ECR プライベートリポジトリの作成
- Amazon ECR でのプライベートリポジトリのコンテンツと詳細の表示
- Amazon ECR でのプライベートリポジトリの削除
- Amazon ECR でのプライベートリポジトリポリシー
- Amazon ECR でのプライベートリポジトリのタグ付け

プライベートリポジトリの概念

- デフォルトでは、アカウントにはデフォルトリポジトリ (*aws_account_id*.dkr.ecr.*region*.amazonaws.com) 内のリポジトリへの読み取りおよび書き込みアクセス権があります。ただし、ユーザーには、Amazon ECR API への呼び出しと、リポジトリに対するイメージのプッシュまたはプルを行う許可が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するための複数の 管理ポリシーが用意されています。詳細については、「Amazon Elastic Container Registry のアイデンティティベースのポリシーの例」を参照してください。
- リポジトリは、 ユーザーアクセスポリシーと個々のリポジトリポリシーによって制御できます。詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。
- リポジトリ名では、似たリポジトリをグループ化するのに使用できる名前空間をサポートできます。たとえば、同じレジストリを使用するチームが複数ある場合、チーム A が team-a 名前空間を使用し、チーム B が team-b 名前を使用することが可能です。こうすることで、各チームは、web-app という独自のイメージを持つことができます。各イメージの先頭には、チームの名前空間が付けられます。この設定により、各チームは干渉することなくイメージを同時に使用でき

ます。チーム A のイメージは `team-a/web-app` で、チーム B のイメージは `team-b/web-app` です。

- イメージは、独自のレジストリ内およびアカウント間でリージョン間で他のリポジトリにレプリケートできます。これを行うには、レジストリ設定でレプリケーション構成を指定します。詳細については、「Amazon ECR でのプライベートレジストリ設定」を参照してください。
- BLOB マウントがレジストリレベルで有効になっている場合、リポジトリは共通のイメージレイヤーを共有できます。

イメージを保存するための Amazon ECR プライベートリポジトリの作成

> **Important**
>
> AWS KMS (DSSE-KMS) を使用した二層式サーバー側の暗号化は、 AWS GovCloud (US) リージョンでのみ使用できます。

Amazon ECR プライベートリポジトリを作成し、そのリポジトリを使用してコンテナイメージを保存します。 AWS マネジメントコンソールを使用してプライベートリポジトリを作成する手順は、次のとおりです。

リポジトリを作成するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、リポジトリを作成するリージョンを選択します。
3. [プライベートリポジトリ] を選択し、[リポジトリの作成] を選択します。
4. [リポジトリ名] に、リポジトリの一意の名前を入力します。リポジトリ名は単独で指定できます (`nginx-web-app` など)。リポジトリ名に名前空間を付けて、リポジトリをカテゴリでグループ化することもできます (`project-a/nginx-web-app` など)。

   > **Note**
   >
   > リポジトリ名には最大 256 文字まで含めることができます。名前は英字で始まる必要があり、小文字、数字、複数のハイフン、複数のアンダースコア、ピリオド、複数のス

ラッシュのみを含めることができます。二重スラッシュの使用はサポートされていません。

5. [Image tag immutability] で、このリポジトリのタグのミュータビリティの設定を選択します。

   - [ミュータブル] – イメージタグを上書きする場合は、このオプションを選択します。Amazon ECR がキャッシュされたイメージを更新できるように、プルスルーキャッシュアクションを使用するリポジトリに推奨されます。さらに、いくつかのミュータブルタグのタグ更新を無効にするには、[Mutable tag exclusion] テキストボックスにタグ名を入力します。複数の類似したタグが一致するワイルドカード (*) を使用できます。
   - [イミュータブル] – イメージタグが上書きされるのを防ぎ、既存のタグを含むイメージをプッシュするときに、リポジトリ内のすべてのタグと除外に適用する場合、このオプションを選択します。Amazon ECR は、既存のタグを使用してイメージをプッシュしようとすると `ImageTagAlreadyExistsException` を返します。さらに、いくつかのイミュータブルタグのタグ更新を有効にするには、[Immutable tag exclusion] テキストボックスにタグ名を入力します。複数の類似したタグが一致するワイルドカード (*) を使用できます。

   > **Note**
   >
   > 個々のタグの可変性設定はサポートされていません。

6. 暗号化設定では、AES-256 または のいずれかを選択します AWS KMS。詳細については、「保管中の暗号化」を参照してください。

   a. を選択した場合 AWS KMS は、シングルレイヤー暗号化とデュアルレイヤー暗号化のどちらかを選択します。 AWS KMS またはデュアルレイヤー暗号化の使用には追加料金がかかります。詳細については、Amazon ECR サービスの料金を参照してください。

   b. デフォルトでは、エイリアスを含む AWS マネージドキー`aws/ecr`が選択されます。このキーは、 AWS KMS 暗号化を有効にしたリポジトリを初めて作成するときにアカウントに作成されます。[カスタマーマネージドキー (高度)] を選択して、独自の AWS KMS キーを選択します。 AWS KMS キーはクラスターと同じリージョンにある必要があります。 AWS KMS キーの作成を選択してコンソール AWS KMS に移動し、独自のキーを作成します。

7. [イメージのスキャン設定] では、基本スキャンに対してリポジトリレベルでのスキャン設定を指定できますが、プライベートレジストリレベルでスキャン設定を指定することがベストプラクティスです。プライベートレジストリレベルでスキャン設定を設定すると、拡張スキャンまたは

基本スキャンのいずれかを選択できます。また、スキャンするリポジトリを指定するフィルターを定義することもできます。

8. [作成] を選択します。

リポジトリを作成するには (AWS CLI)

1. AWS CLI aws ecr create-repository コマンドで を使用してリポジトリを作成できます。

```
aws ecr create-repository \
            --repository-name hello-repository \
            --region region
```

2. リポジトリ作成テンプレートが定義されている場合は、使い慣れた Amazon ECR プッシュコマンドと目的のリポジトリ名を使用してイメージをプッシュすることで、リポジトリを作成できます。Amazon ECR は、リポジトリ作成テンプレートの事前定義された設定を使用して、自動的にリポジトリを作成します。リポジトリ作成テンプレートがまだ定義されていない場合、存在しないイメージリポジトリへのリクエストは失敗します。

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/prefix/my-new-
repository:tag
```

次の手順

イメージをリポジトリにプッシュする手順を表示するには、リポジトリを選択し、[プッシュコマンドの表示] を選択します。リポジトリへのイメージのプッシュの詳細については、「Amazon ECR プライベートリポジトリへのイメージのプッシュ」を参照してください。

Amazon ECR でのプライベートリポジトリのコンテンツと詳細の表示

プライベートリポジトリを作成したら、 AWS マネジメントコンソールでリポジトリに関する詳細を表示できます。

- リポジトリに保存されているイメージ
- リポジトリに格納されている各イメージの詳細 (各イメージのサイズと SHA ダイジェストを含む)
- リポジトリのコンテンツに指定されたスキャン頻度

- リポジトリにアクティブなプルスルーキャッシュルールが関連付けられているかどうか
- リポジトリの暗号化設定

> **Note**
>
> Docker バージョン 1.9 以降、Docker クライアントは V2 Docker レジストリにプッシュする前にイメージレイヤーを圧縮します。docker images コマンドの出力は、圧縮されていないイメージのサイズを表示します。したがって、Docker は、 AWS マネジメントコンソールで表示されているイメージよりも大きなイメージを返す可能性があることに注意してください。

リポジトリ情報を表示するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、表示するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] (リポジトリ) ページで、[Private] (プライベート) タブを選択し、表示するレポジトリを選択します。
5. リポジトリの詳細ページでは、コンソールはデフォルトで [Images] (イメージ) ビューを表示します。ナビゲーションメニューを使用して、リポジトリに関するその他の情報を表示します。
   - [Summary] (概要) をクリックして、リポジトリの詳細とリポジトリのプルカウントデータを表示します。
   - [Images] (イメージ) を選択して、リポジトリ内のイメージタグに関する情報を表示します。イメージの詳細情報を表示するには、イメージタグを選択します。詳細については、「Amazon ECR でのイメージの詳細の表示」を参照してください。

     削除するイメージのうちタグが付いていないものがある場合、削除するリポジトリの左側のボックスを選択し、[Delete] (削除) を選択できます。詳細については、「Amazon ECR でのイメージの削除」を参照してください。
   - [Permissions (アクセス許可)] を選択し、リポジトリに適用されるリポジトリポリシーを表示します。詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。
   - [ライフサイクルポリシー] を選択し、リポジトリに適用されるライフサイクルポリシールールを表示します。ライフサイクルイベント履歴もここに表示されます。詳細については、

「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。

- [タグ] を選択して、リポジトリに適用されているメタデータタグを表示します。

Amazon ECR でのプライベートリポジトリの削除

使用し終わったリポジトリは削除できます。でリポジトリを削除すると AWS マネジメントコンソール、リポジトリに含まれるすべてのイメージも削除されます。これは元に戻すことはできません。

> **Important**
>
> 削除されたリポジトリ内のイメージもすべて削除されます。このオペレーションは元に戻すことができません。

リポジトリを削除するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、削除するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] (リポジトリ) ページで、[Private] (プライベート) タブを選択し、削除するレポジトリを選択して [Delete] (削除) を選びます。
5. [***repository_name*** の削除] ウィンドウで、選択されたリポジトリを削除することを確認し、[削除] を選択します。

Amazon ECR でのプライベートリポジトリポリシー

Amazon ECR では、リソースベースのアクセス権限を使用してリポジトリへのアクセスを制御します。リソースベースのアクセス許可を使用すると、リポジトリにアクセスできるユーザーやロール、およびそれらのユーザーやロールがリポジトリに対して実行できるアクションを指定できます。デフォルトでは、リポジトリを作成した AWS アカウントのみがリポジトリにアクセスできます。リポジトリに対する追加アクセスを許可するリポジトリポリシーを適用できます。

トピック

- レポジトリポリシーと IAM ポリシー

- Amazon ECR でのプライベートリポジトリポリシーの例
- Amazon ECR でのプライベートリポジトリポリシーステートメントの設定

レポジトリポリシーと IAM ポリシー

Amazon ECR リポジトリポリシーは、個別の Amazon ECR リポジトリへのアクセスを制御することを目的として特に使用される IAM ポリシーのサブセットです。IAM ポリシーは、一般的に Amazon ECR サービス全体にアクセス権限を適用するために使用されますが、特定のリソースへのアクセスを制限するために使用することもできます。

Amazon ECR リポジトリポリシーと IAM ポリシーはどちらも、特定のユーザーまたはロールがリポジトリで実行できるアクションを決定するときに使用されます。ユーザーまたはロールがレポジトリから 1 つのアクションの実行を許可されていても、IAM ポリシーからアクセス権限を拒否される場合 (またはその逆の場合)、そのアクションは拒否されます。ユーザーあるいはロールは、レポジトリポリシーまたは IAM ポリシーのいずれかのみでアクションへのアクセスが許可されていることを必要とし、その両方でこのアクションが許可されている必要はありません。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリへの認証を行って、Amazon ECR リポジトリに対するイメージのプッシュまたはプルを行う前に、IAM ポリシーを介して `ecr:GetAuthorizationToken` API への呼び出しを行う許可が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するための複数の IAM マネージドポリシーが用意されています。詳細については、「Amazon Elastic Container Registry のアイデンティティベースのポリシーの例」を参照してください。

これらのいずれかのポリシータイプを使用して、次の例に示すようにレポジトリへのアクセスを制御します。

この例は、特定のユーザーがリポジトリ内でリポジトリとイメージを説明することを許可する Amazon ECR リポジトリポリシーを示しています。

JSON

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

```
    "Statement": [
        {
            "Sid": "ECRRepositoryPolicy",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::111122223333:user/username"},
            "Action": [
                "ecr:DescribeImages",
                "ecr:DescribeRepositories"
            ],
            "Resource": "*"
        }
    ]
}
```

この例は、リソースパラメータを使用してポリシーで 1 つのレポジトリ (レポジトリの完全な ARN で指定) を対象とすることで、上記と同じ目的を達成する IAM ポリシーを示しています。Amazon リソースネーム (ARN) 形式の詳細については、「リソース」を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDescribeRepoImage",
            "Effect": "Allow",
            "Action": [
                "ecr:DescribeImages",
                "ecr:DescribeRepositories"
            ],
            "Resource": ["arn:aws:ecr:us-
east-1:111122223333:repository/repository-name"]
        }
    ]
}
```

Amazon ECR でのプライベートリポジトリポリシーの例

> **Important**
>
> このページのリポジトリポリシーの例は、Amazon ECR プライベートリポジトリに適用するためのものです。Amazon ECR リポジトリをリソースとして指定するように変更しない限り、IAM プリンシパルと直接使用すると正しく動作しません。リポジトリポリシーの設定詳細については、「Amazon ECR でのプライベートリポジトリポリシーステートメントの設定」を参照してください。

Amazon ECR リポジトリポリシーは、個別の Amazon ECR リポジトリへのアクセスを制御することを目的として特に使用される IAM ポリシーのサブセットです。IAM ポリシーは、一般的に Amazon ECR サービス全体にアクセス権限を適用するために使用されますが、特定のリソースへのアクセスを制限するために使用することもできます。詳細については、「レポジトリポリシーと IAM ポリシー」を参照してください。

以下のリポジトリポリシーの例では、Amazon ECR プライベートリポジトリへのアクセスを制御するために使用できる許可ステートメントを示しています。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリへの認証を行って、Amazon ECR リポジトリに対するイメージのプッシュまたはプルを行う前に、IAM ポリシーを介して `ecr:GetAuthorizationToken` API への呼び出しを行う許可が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するための複数の IAM マネージドポリシーが用意されています。詳細については、「Amazon Elastic Container Registry のアイデンティティベースのポリシーの例」を参照してください。

例: 1 名以上の ユーザーを許可する

次のリポジトリポリシーでは、1 名以上の ユーザーがリポジトリに対してイメージをプッシュまたはプルできます。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPushPull",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/push-pull-user-1",
                    "arn:aws:iam::111122223333:user/push-pull-user-2"
                ]
            },
            "Action": [
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:GetDownloadUrlForLayer",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ],
            "Resource": "*"
        }
    ]
}
```

例: 別のアカウントを許可する

次のリポジトリポリシーでは、特定のアカウントにイメージのプッシュが許可されます。

> **Important**
>
> アクセス権限を付与するアカウントには、リポジトリポリシーを作成するリージョンが有効になっている必要があります。有効でない場合、エラーが発生します。

JSON

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

```
            "Sid": "AllowCrossAccountPush",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ],
            "Resource": "*"
        }
    ]
}
```

次のリポジトリポリシーでは、一部のユーザーにイメージのプルが許可されますが (*pull-user-1* および *pull-user-2*)、別のユーザーにはフルアクセスが許可されます (*admin-user*)。

> **Note**
>
> で現在サポートされていないより複雑なリポジトリポリシーについては AWS マネジメントコンソール、 set-repository-policy AWS CLI コマンドを使用してポリシーを適用できます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPull",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/pull-user-1",
                    "arn:aws:iam::111122223333:user/pull-user-2"
                ]
            },
            "Action": [
```

```
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowAll",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:user/admin-user"
            },
            "Action": [
                "ecr:*"
            ],
            "Resource": "*"
        }
    ]
}
```

例: すべて拒否する

次のリポジトリポリシーでは、すべてのアカウントのすべてのユーザーに対してイメージのプルが拒否されます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DenyPull",
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*"
        }
    ]
}
```

例: アクセスを特定の IP アドレスに制限する

次の例では、特定のアドレス範囲からリポジトリに適用された場合に、Amazon ECR オペレーションを実行するアクセス許可をすべてのユーザーに対して拒否します。

このステートメントの条件では、許可されたインターネットプロトコルバージョン 4 (IPv4) IP アドレスの `54.240.143.*` 範囲を識別します。

`Condition` ブロックは、 `NotIpAddress` 条件と `aws:SourceIp`条件キーを使用します。これは、AWS全体の条件キーです。これらの条件キーの詳細については、「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。`aws:sourceIp`IPv4 値は標準の CIDR 表記を使用します。詳細については、IAM ユーザーガイドの IP アドレス条件演算子 を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Id": "ECRPolicyId1",
    "Statement": [
        {
            "Sid": "IPAllow",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "ecr:*",
            "Resource": "*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "54.240.143.0/24"
                }
            }
        }
    ]
}
```

例: AWS サービスを許可する

次のリポジトリポリシーは、そのサービスとの統合に必要な Amazon ECR API アクション AWS CodeBuild へのアクセスを許可します。次の例を使用する場合、`aws:SourceArn` 条件キーと `aws:SourceAccount` 条件キーを使用して、これらのアクセス許可を引き受けることができる

リソースの範囲を指定する必要があります。詳細については、AWS CodeBuild ユーザーガイドの「CodeBuild の Amazon ECR サンプル」を参照してください。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CodeBuildAccess",
         "Effect":"Allow",
         "Principal":{
            "Service":"codebuild.amazonaws.com"
         },
         "Action":[
            "ecr:BatchGetImage",
            "ecr:GetDownloadUrlForLayer"
         ],
         "Resource": "*",
         "Condition":{
            "ArnLike":{
               "aws:SourceArn":"arn:aws:codebuild:us-
east-1:123456789012:project/project-name"
            },
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            }
         }
      }
   ]
}
```

Amazon ECR でのプライベートリポジトリポリシーステートメントの設定

以下の AWS マネジメントコンソール 手順に従って、 のリポジトリにアクセスポリシーステートメントを追加できます。リポジトリごとに複数のポリシーステートメントを追加できます。エンドポイントポリシーの例については、「Amazon ECR でのプライベートリポジトリポリシーの例」を参照してください。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリへの認証を行って、Amazon ECR リポジトリに対するイメージのプッシュまたはプルを行う前に、IAM ポリシーを介して `ecr:GetAuthorizationToken` API への呼び出しを行う許可が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するための複数の IAM マネージドポリシーが用意されています。詳細については、「Amazon Elastic Container Registry のアイデンティティベースのポリシーの例」を参照してください。

リポジトリポリシーステートメントを設定するには

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、ポリシーステートメントをオンに設定するリポジトリが含まれるリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、ポリシーステートメントを有効に設定するリポジトリを選択して、リポジトリの内容を表示します。
5. リポジトリイメージのリストビューのナビゲーションペインで、[アクセス権限]、[編集] を選択します。

   > **Note**
   >
   > ナビゲーションペインに [Permissions] (アクセス許可) オプションが表示されない場合、画面がリポジトリイメージリストビューであることを確認します。

6. 許可を編集ページで [ステートメントを追加] を選択します。
7. [ステートメント名] に、ステートメントの名前を入力します。
8. [効果] で、ポリシーステートメントの結果を許可または明示的な拒否のどちらにするかを指定します。
9. [プリンシパル] で、ポリシーステートメントを適用する範囲を選択します。詳細については、IAM ユーザーガイドの「AWS JSON ポリシー要素: プリンシパル」を参照してください。
   - Everyone (*) チェックボックスをオンにすると、認証されたすべての AWS ユーザーにステートメントを適用できます。

- [サービスプリンシパル] で、特定のサービスにステートメントを適用するサービスプリンシパル名 (ecs.amazonaws.com など) を指定します。
- AWS アカウント IDs では、 AWS アカウント番号 (など111122223333) を指定して、特定の AWS アカウントのすべてのユーザーにステートメントを適用します。カンマ区切りのリストを使用して複数のアカウントを指定できます。

> **Important**
>
> アクセス権限を付与するアカウントには、リポジトリポリシーを作成するリージョンが有効になっている必要があります。有効でない場合、エラーが発生します。

- IAM エンティティの場合、ステートメントを適用する AWS アカウントのロールまたはユーザーを選択します。

> **Note**
>
> で現在サポートされていないより複雑なリポジトリポリシーについては AWS マネジメントコンソール、 set-repository-policy AWS CLI コマンドを使用してポリシーを適用できます。

10. [アクション] で、個別の API オペレーションのリストとの間でポリシーステートメントを適用する Amazon ECR API オペレーションの範囲を選択します。
11. 完了したら、[Save] を選択してポリシーを設定します。
12. 追加する各レポジトリポリシーに対して、前のステップを繰り返します。

Amazon ECR でのプライベートリポジトリのタグ付け

Amazon ECR リポジトリの管理に役立つように、 AWS リソースタグを使用して、新規または既存の Amazon ECR リポジトリに独自のメタデータを割り当てることができます。たとえば、アカウントの Amazon ECR リポジトリに対して各リポジトリの所有者を追跡しやすくするため、一連のタグを定義できます。

タグの基本

タグには、Amazon ECR に関連する意味はなく、完全に文字列として解釈されます。タグは自動的にはリソースに割り当てられません。タグキーと値は編集でき、リソースからタグはいつでも削除できます。タグの値を空の文字列に設定することはできますが、タグの値を null に設定することはで

きません。特定のリソースについて既存のタグと同じキーを持つタグを追加した場合、以前の値は新しい値によって上書きされます。リソースを削除すると、リソースのタグもすべて削除されます。

Amazon ECR コンソール、、 AWS CLIおよび Amazon ECR API を使用してタグを操作できます。

AWS Identity and Access Management (IAM) を使用すると、 AWS アカウント内のどのユーザーにタグを作成、編集、または削除するためのアクセス許可があるかを制御できます。IAM ポリシー内のタグの詳細については、「the section called “タグベースのアクセスコントロールを使用する”」を参照してください。

請求用のリソースにタグを付ける

Amazon ECR リポジトリに追加するタグは、コスト配分を有効にした後にコストと使用状況レポートでコスト配分を確認するときに便利です。詳しくは、「Amazon ECR 使用状況レポート」を参照してください。

リソースを組み合わせたコストを確認するには同じタグキー値を持つリソースに基づいて、請求情報を整理します。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することで、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。タグによるコスト配分レポートの設定の詳細については、AWS Billing ユーザーガイドの「毎月のコスト配分レポート」を参照してください。

> **Note**
>
> レポートを有効にすると、約 24 時間後に、今月のデータを表示できるようになります。

Amazon ECR でのプライベートリポジトリへのタグの追加

プライベートリポジトリにタグを追加できます。

タグの名前とベストプラクティスの詳細については、「 リソースのタグ付けユーザーガイド」の「タグの命名制限と要件」と「ベストプラクティス」を参照してください。 AWS

リポジトリへのタグの追加 (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。

4. [リポジトリ] ページで、タグ付けするリポジトリの横にあるチェックボックスを選択します。
5. [アクション] メニューから、[リポジトリタグ] を選択します。
6. [リポジトリタグ] ページで、[タグの追加]、[タグの追加] の順に選択します。
7. [リポジトリタグを編集] ページで、各タグのキーと値を指定してから [保存] を選択します。

リポジトリ (AWS CLI または API) へのタグの追加

または API を使用して、1 つ以上のタグを追加 AWS CLI または上書きできます。

- AWS CLI - tag-resource
- API アクション - TagResource

次の例は、 AWS CLIを使用してタグを追加する方法を示しています。

例 1: リポジトリにタグ付けする

次のコマンドは、リポジトリにタグ付けします。

```
aws ecr tag-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name \
     --tags Key=stack,Value=dev
```

例 2: リポジトリに複数のタグをタグ付けする

次のコマンドでは、リポジトリに 3 つのタグが追加されます。

```
aws ecr tag-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name \
     --tags Key=key1,Value=value1 Key=key2,Value=value2 Key=key3,Value=value3
```

例 3: リポジトリのタグを一覧表示する

次のコマンドは、リポジトリに関連付けられているタグをリストします。

```
aws ecr list-tags-for-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name
```

例 4: リポジトリを作成してタグを追加する

次のコマンドでは、test-repo という名前のリポジトリを作成し、キーが team で値が devs のタグを追加します。

```
aws ecr create-repository \
     --repository-name test-repo \
     --tags Key=team,Value=devs
```

Amazon ECR でのプライベートリポジトリからのタグの削除

プライベートリポジトリからタグを削除できます。

プライベートリポジトリからタグを削除する手順 (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. [リポジトリ] ページで、タグを削除したいリポジトリの横にあるチェックボックスを選択します。
4. [アクション] メニューから、[リポジトリタグ] を選択します。
5. [リポジトリタグ] ページで、[編集] を選択します。
6. [リポジトリタグを編集] ページで、削除するタグごとに [削除] を選択してから [保存] を選択します。

プライベートリポジトリからタグを削除する手順 (AWS CLI)

または AWS CLI API を使用して、1 つ以上のタグを削除できます。

- AWS CLI - untag-resource
- API アクション - UntagResource

次の例は、 AWS CLIを使用してリポジトリからタグを削除する方法を示しています。

```
aws ecr untag-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name \
     --tag-keys tag_key
```

Amazon ECR のプライベートイメージ

Amazon ECR は、Docker イメージ、Open Container Initiative (OCI) イメージ、OCI 互換アーティファクトをプライベートリポジトリに保存します。Docker CLI またはその他のクライアントを使用して、リポジトリからイメージをプッシュおよびプルできます。

Amazon ECR での OCI v1.1 のサポートにより、OCI Referrers API で定義されるリファレンスアーティファクトを保存および管理できます。アーティファクトには、署名、ソフトウェア部品表 (SBoM)、Helm チャート、スキャン結果、認証が含まれます。コンテナイメージの一連のアーティファクトは、そのコンテナと一緒に転送され、別のイメージとして保存されます。このイメージはリポジトリの消費量としてカウントされます。

「Amazon ECR でイメージに署名する」および「Amazon ECR プライベートリポジトリからの署名およびその他のアーティファクトの削除」ページでは、署名関連のアーティファクトを使用する方法の例が示されています。コンテナイメージの署名の詳細については、「AWS Signer デベロッパーガイド」の「Signing container images」を参照してください。

トピック

- Amazon ECR プライベートリポジトリへのイメージのプッシュ
- Amazon ECR プライベートリポジトリからの署名およびその他のアーティファクトの削除
- Amazon ECR でのイメージの詳細の表示
- Amazon ECR プライベートリポジトリからローカル環境へのイメージのプル
- Amazon Linux コンテナイメージのプル
- Amazon ECR でのイメージの削除
- Amazon ECR でのイメージのアーカイブ
- Amazon ECR でのイメージの再タグ付け
- Amazon ECR でのイメージタグの上書きの防止
- Amazon ECR でのコンテナイメージマニフェスト形式のサポート
- Amazon ECS での Amazon ECR イメージの使用
- Amazon EKS での Amazon ECR イメージの使用

Amazon ECR プライベートリポジトリへのイメージのプッシュ

Docker イメージ、マニフェストリスト、Open Container Initiative (OCI) イメージと互換アーティファクトをプライベートリポジトリにプッシュできます。

Amazon ECR は、イメージを他のリポジトリにレプリケートする方法を提供します。プライベートレジストリ設定でレプリケーション設定を指定することで、自分のレジストリ内のリージョン間や、異なるアカウント間でレプリケートすることができます。詳細については、「Amazon ECR でのプライベートレジストリ設定」を参照してください。

> **Note**
>
> 現在アーカイブされているイメージをプッシュすると、そのイメージは自動的に復元され、アーカイブから削除されます。イメージのアーカイブと復元の詳細については、「」を参照してくださいAmazon ECR でのイメージのアーカイブ。

レジストリ BLOB マウントが有効で、マウントパラメータが含まれている場合、Amazon ECR はプッシュオペレーション中にレジストリ内の既存のレイヤーを自動的にチェックします。同じレジストリ内の別のリポジトリにレイヤーがすでに存在する場合、Amazon ECR は重複をアップロードする代わりに既存のレイヤーをマウントします。詳細については、「Amazon ECR での Blob マウント」を参照してください。

トピック

- Amazon ECR プライベートリポジトリにイメージをプッシュするための IAM アクセス許可
- Amazon ECR プライベートリポジトリへの Docker イメージのプッシュ
- Amazon ECR プライベートリポジトリへのマルチアーキテクチャイメージのプッシュ
- Amazon ECR プライベートリポジトリへの Helm チャートのプッシュ

Amazon ECR プライベートリポジトリにイメージをプッシュするための IAM アクセス許可

ユーザーには、Amazon ECR プライベートリポジトリにイメージをプッシュするための IAM アクセス許可が必要です。最小特権の付与に関するベストプラクティスに従って、特定のリポジトリへのアクセスを許可できます。また、すべてのリポジトリへのアクセスを許可することもできます。

ユーザーは、認可トークンをリクエストして、イメージをプッシュする各 Amazon ECR レジストリに対して認証を受ける必要があります。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するための AWS 管理ポリシーがいくつか用意されています。詳細については、「AWS Amazon Elastic Container Registry の マネージドポリシー」を参照してください。

独自の IAM ポリシーを作成することもできます。次の IAM ポリシーは、特定のリポジトリにイメージをプッシュするために必要なアクセス許可を付与します。特定のリポジトリのアクセス許可を制限するには、リポジトリの完全な Amazon リソースネーム (ARN) を使用します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:CompleteLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:InitiateLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:PutImage",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:us-
east-1:111122223333:repository/repository-name"
        },
        {
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

次の IAM ポリシーは、すべてのリポジトリにイメージをプッシュするために必要なアクセス許可を付与します。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                    "ecr:CompleteLayerUpload",
                    "ecr:GetAuthorizationToken",
                    "ecr:UploadLayerPart",
                    "ecr:InitiateLayerUpload",
                    "ecr:BatchCheckLayerAvailability",
                    "ecr:PutImage",
                    "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:us-west-2:111122223333:repository/*"
        }
    ]
}
```

Amazon ECR プライベートリポジトリへの Docker イメージのプッシュ

docker pushコマンドを使用してコンテナイメージを Amazon ECR リポジトリにプッシュできます。

Amazon ECR では、マルチアーキテクチャイメージに使用される Docker マニフェストリストの作成とプッシュもサポートされています。詳細については、「Amazon ECR プライベートリポジトリへのマルチアーキテクチャイメージのプッシュ」を参照してください。

Docker イメージを Amazon ECR リポジトリにプッシュするには

イメージをプッシュする前に Amazon ECR リポジトリが存在するか、リポジトリ作成テンプレートが定義されている必要があります。詳細については、「イメージを保存するための Amazon ECR プライベートリポジトリの作成」および「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。

1. イメージのプッシュ先となる Amazon ECR レジストリに対して Docker クライアントを認証します。認証トークンは、使用するレジストリごとに取得する必要があり、トークンは 12 時間有効です。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。

   Amazon ECR レジストリに対して Docker を認証するには、aws ecr get-login-password コマンドを実行します。認証トークンを docker login コマンドに渡すとき、ユーザー名の AWS 値を使

用し、認証先の Amazon ECR レジストリの URI を指定します。複数のレジストリに対して認証する場合は、レジストリごとにコマンドを繰り返す必要があります。

> **Important**
>
> エラーが発生した場合は、 AWS CLIの最新バージョンをインストールまたはアップグレードします。詳細については、「AWS Command Line Interface ユーザーガイド」の「Installing the AWS Command Line Interface」を参照してください。

```
aws ecr get-login-password --region <region> | docker login --username AWS --
password-stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com
```

2. イメージリポジトリがまだプッシュ先のレジストリに存在しておらず、リポジトリ作成テンプレートが定義されている場合は、リポジトリ作成テンプレートのプレフィックスと目的のリポジトリ名を使用してイメージをプッシュできます。ECR は、リポジトリ作成テンプレートの事前定義された設定を使用して、自動的にリポジトリを作成します。

   一致するリポジトリ作成テンプレートが定義されていない場合は、リポジトリを作成する必要があります。詳細については、プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレートまたはイメージを保存するための Amazon ECR プライベートリポジトリの作成を参照してください。

3. プッシュするローカルイメージを識別します。docker images コマンドを実行し、システム上のコンテナイメージを一覧表示します。

   ```
   docker images
   ```

   イメージは、結果のコマンド出力で *repository:tag* の値またはイメージ ID によって識別できます。

4. Amazon ECR レジストリ、リポジトリ、およびオプションのイメージタグ名を組み合わせたタグをイメージに付与します。レジストリ形式は *aws_account_id*.dkr.ecr.*region*.amazonaws.comです。リポジトリ名は、イメージ用に作成したリポジトリと一致する必要があります。イメージタグを省略した場合、タグは latest と見なされます。

   次の例では、ローカルイメージに ID *e9ae3c220b23* を としてタグ付けしますaws_account_id.dkr.ecr.region.amazonaws.com/my-repository:tag。

```
docker tag e9ae3c220b23 aws_account_id.dkr.ecr.region.amazonaws.com/my-
repository:tag
```

5. docker push コマンドを使用してイメージをプッシュします。

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:tag
```

6. (オプション) 追加のタグをイメージに適用し、Step 4 とStep 5 を繰り返して、それらのタグを Amazon ECR にプッシュします。

Amazon ECR プライベートリポジトリへのマルチアーキテクチャイメージのプッシュ

Docker マニフェストリストを作成してプッシュすることで、マルチアーキテクチャイメージを Amazon ECR リポジトリにプッシュできます。マニフェストリストは、1 つ以上のイメージ名を指定して作成されるイメージのリストです。ほとんどの場合、マニフェストリストは、同じ機能を提供するが異なるオペレーティングシステムまたはアーキテクチャ向けのイメージから作成されます。マニフェストリストは必須ではありません。詳細については、「Docker マニフェスト」を参照してください。

マニフェストリストは、他の Amazon ECR イメージと同様に、Amazon ECS タスク定義または Amazon EKS ポッド仕様でプルまたは参照できます。

前提条件

- Docker CLI で実験機能をオンにします。実験機能の詳細については、Docker ドキュメントの「Experimental features」を参照してください。
- イメージをプッシュする前に、Amazon ECR リポジトリが存在している必要があります。詳細については、「the section called “イメージを保存するためのリポジトリの作成”」を参照してください。
- Docker マニフェストを作成する前に、イメージをリポジトリにプッシュする必要があります。イメージをプッシュする方法については、「Amazon ECR プライベートリポジトリへの Docker イメージのプッシュ」を参照してください。

マルチアーキテクチャ Docker イメージを Amazon ECR リポジトリにプッシュするには

1. イメージのプッシュ先となる Amazon ECR レジストリに対して Docker クライアントを認証します。認証トークンは、使用するレジストリごとに取得する必要があり、トークンは 12 時間有効です。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。

   Amazon ECR レジストリに対して Docker を認証するには、aws ecr get-login-password コマンドを実行します。認証トークンを docker login コマンドに渡すとき、ユーザー名の AWS 値を使用し、認証先の Amazon ECR レジストリの URI を指定します。複数のレジストリに対して認証する場合は、レジストリごとにコマンドを繰り返す必要があります。

   > **Important**
   >
   > エラーが発生した場合は、 AWS CLIの最新バージョンをインストールまたはアップグレードします。詳細については、「AWS Command Line Interface ユーザーガイド」の「Installing the AWS Command Line Interface」を参照してください。

   ```
   aws ecr get-login-password --region <region> | docker login --username AWS --
   password-stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com
   ```

2. リポジトリ内のイメージをリストし、イメージタグを確認します。

   ```
   aws ecr describe-images --repository-name my-repository
   ```

3. Docker マニフェストリストを作成します。manifest create コマンドは、参照されたイメージがリポジトリにすでに存在することを確認し、マニフェストをローカルに作成します。

   ```
   docker manifest create aws_account_id.dkr.ecr.region.amazonaws.com/my-
   repository aws_account_id.dkr.ecr.region.amazonaws.com/my-
   repository:image_one_tag aws_account_id.dkr.ecr.region.amazonaws.com/my-
   repository:image_two
   ```

4. (オプション) Docker マニフェストリストを検査します。これにより、マニフェストリストで参照される各イメージマニフェストのサイズとダイジェストを確認できます。

   ```
   docker manifest inspect aws_account_id.dkr.ecr.region.amazonaws.com/my-repository
   ```

5. Docker マニフェストリストを Amazon ECR リポジトリにプッシュします。

```
docker manifest push aws_account_id.dkr.ecr.region.amazonaws.com/my-repository
```

Amazon ECR プライベートリポジトリへの Helm チャートのプッシュ

Open Container Initiative (OCI) アーティファクトを Amazon ECR リポジトリにプッシュできます。この機能の例を確認するには、次の手順を使用して Helm チャートを Amazon ECR にプッシュします。

Amazon ECR でホストされる Helm チャートを Amazon EKS で使用する方法については、「Amazon EKS クラスターに Helm チャートをインストールする」を参照してください。

Helm チャートを Amazon ECR リポジトリにプッシュするには

1. Helm クライアントの最新バージョンをインストールします。これらのステップは、Helm バージョン 3.18.6 を使用して作成されました。Amazon EKS でサポートされている Kubernetes バージョンとの互換性を維持するには、Helm バージョン 3.9 以降を使用します。詳細については、「Installing Helm」を参照してください。
2. 次の手順に従って、テスト Helm チャートを作成します。詳細については、Helm ドキュメントの「Getting Started」を参照してください。

   a. `helm-test-chart` という名前の Helm チャートを作成し、`templates` ディレクトリの内容をクリアします。

   ```
   helm create helm-test-chart
   rm -rf ./helm-test-chart/templates/*
   ```

   b. `templates` フォルダに `ConfigMap` を作成します。

   ```
   cd helm-test-chart/templates
   cat <<EOF > configmap.yaml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: helm-test-chart-configmap
   data:
     myvalue: "Hello World"
   EOF
   ```

3. チャートをパッケージ化します。出力には、Helm チャートをプッシュするときに使用する、パッケージ化されたチャートのファイル名が含まれます。

```
cd ../..
helm package helm-test-chart
```

Output

```
Successfully packaged chart and saved it to: /Users/username/helm-test-chart-0.1.0.tgz
```

4. Helm チャートを格納するリポジトリを作成します。リポジトリの名前は、ステップ 2 で Helm チャートを作成する場合に使用する名前と一致する必要があります。詳しくは、「イメージを保存するための Amazon ECR プライベートリポジトリの作成」を参照してください。

```
aws ecr create-repository \
     --repository-name helm-test-chart \
     --region us-west-2
```

5. Helm チャートのプッシュ先となる Amazon ECR レジストリに対して Helm クライアントを認証します。認証トークンは、使用するレジストリごとに取得する必要があり、トークンは 12 時間有効です。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。

```
aws ecr get-login-password \
     --region us-west-2 | helm registry login \
     --username AWS \
     --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

6. helm push コマンドを使用して Helm チャートをプッシュします。出力には、Amazon ECR リポジトリ URI と SHA ダイジェストが含まれているはずです。

```
helm push helm-test-chart-0.1.0.tgz
 oci://aws_account_id.dkr.ecr.region.amazonaws.com/
```

7. Helm チャートを説明します。

```
aws ecr describe-images \
     --repository-name helm-test-chart \
     --region us-west-2
```

出力で、artifactMediaType パラメータが適切なアーティファクトタイプを示していることを確認します。

```
{
    "imageDetails": [
        {
            "registryId": "aws_account_id",
            "repositoryName": "helm-test-chart",
            "imageDigest":
 "sha256:dd8aebdda7df991a0ffe0b3d6c0cf315fd582cd26f9755a347a52adEXAMPLE",
            "imageTags": [
                "0.1.0"
            ],
            "imageSizeInBytes": 1620,
            "imagePushedAt": "2021-09-23T11:39:30-05:00",
            "imageManifestMediaType": "application/vnd.oci.image.manifest.v1+json",
            "artifactMediaType": "application/vnd.cncf.helm.config.v1+json"
        }
    ]
}
```

8. (オプション) 追加のステップについては、Helm ConfigMap をインストールして Amazon EKS の使用を開始します。詳細については、「Amazon EKS クラスターに Helm チャートをインストールする」を参照してください。

Amazon ECR プライベートリポジトリからの署名およびその他のアーティファクトの削除

ORAS クライアントを使用して、Amazon ECR プライベートリポジトリの署名およびその他のリファレンスタイプのアーティファクトをリストしたり、削除したりできます。署名およびその他のリファレンスアーティファクトの削除は、イメージの削除方法と似ています (「Amazon ECR でのイメージの削除」を参照)。アーティファクトをリストして署名を削除する方法は次のとおりです。

ORAS CLI を使用してイメージアーティファクトを管理する手順

1. ORAS クライアントをインストールして設定します。

   ORAS クライアントをインストールして設定する方法については、ORAS ドキュメントの「Installation」を参照してください。

2. Amazon ECR イメージの使用可能なアーティファクトをリストするには、oras discover を使用し、それに続けてイメージ名を指定します。

```
oras discover 111222333444.dkr.ecr.us-east-1.amazonaws.com/oci:helloworld
```

出力は次のようになります。

```
111222333444.dkr.ecr.us-east-1.amazonaws.com/
oci@sha256:88c0c54329bfdc1d94d6f58cd3fcb1226d46f58670f44a8c689cb3c9b37b6925
### application/vnd.cncf.notary.signature
    ### sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
    ### sha256:6527bcec87adf1d55460666183b9d0968b3cd4e4bc34602d485206a219851171
```

3. ORAS CLI を使用して署名を削除するには、前の例を参考にして、次のコマンドを実行します。

```
oras manifest delete 111222333444.dkr.ecr.us-east-1.amazonaws.com/
oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
```

出力は次のようになります。

```
Are you sure you want to delete the manifest "111222333444.dkr.ecr.us-
east-1.amazonaws.com/
oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42" and
 all tags associated with it? [y/N] y
```

4. y キーを押します。アーティファクトを削除する必要があります。

アーティファクトの削除をトラブルシューティングする手順

先ほど示したような署名の削除が失敗した場合、次のような出力が表示されます。

```
Error response from registry: failed to delete 111222333444.dkr.ecr.us-
east-1.amazonaws.com/
oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42:
unsupported: Requested image referenced by manifest list:
 [sha256:005e2c97a6373e483799fa4ff29ac64a42dd10f08efcc166d6775f9b74943b5b]
```

この失敗は、OCI 1.1 の使用開始前にプッシュされたイメージを削除した場合に発生する可能性があります。エラーで示されているように、イメージを削除する前に、そのイメージを参照しているマニフェストを削除する必要があります。これを行うには、次の手順を実行します。

1. 削除する署名に関連付けられているマニフェストを削除するには、次のように入力します。

```
 oras manifest delete 111222333444.dkr.ecr.us-east-1.amazonaws.com/
oci@sha256:005e2c97a6373e483799fa4ff29ac64a42dd10f08efcc166d6775f9b74943b5b
```

   出力は次のようになります。

```
Are you sure you want to delete the manifest
 "sha256:005e2c97a6373e483799fa4ff29ac64a42dd10f08efcc166d6775f9b74943b5b" and all
 tags associated with it? [y/N] y
```

2. y キーを押します。マニフェストを削除する必要があります。
3. マニフェストがなくなったら、署名を削除できます。

```
 oras manifest delete 111222333444.dkr.ecr.us-east-1.amazonaws.com/
oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
```

   出力は次のようになります。y キーを押します。

```
Are you sure you want to delete the manifest
 "sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42" and all
 tags associated with it? [y/N] y
Deleted [registry] 111222333444.dkr.ecr.us-east-1.amazonaws.com/
oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
```

4. 署名が削除されたことを確認するには、次のように入力します。

```
oras discover 111222333444.dkr.ecr.us-east-1.amazonaws.com/oci:helloworld
```

   出力は次のようになります。

```
111222333444.dkr.ecr.us-east-1.amazonaws.com/
oci@sha256:88c0c54329bfdc1d94d6f58cd3fcb1226d46f58670f44a8c689cb3c9b37b6925
### application/vnd.cncf.notary.signature
    ### sha256:6527bcec87adf1d55460666183b9d0968b3cd4e4bc34602d485206a219851171
```

Amazon ECR でのイメージの詳細の表示

イメージをリポジトリにプッシュした後、そのイメージに関する情報を表示できます。詳細には、以下の情報が含まれます。

- イメージ URI
- イメージフラグ
- アーティファクトメディアのタイプ
- イメージマニフェストのタイプ
- スキャンステータス
- イメージのサイズ (MB)
- イメージがいつリポジトリにプッシュされたか
- レプリケーションステータス

イメージの詳細を表示するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、イメージを含むリポジトリが含まれるリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、表示するリポジトリを選択します。
5. リポジトリ: ***repository_name*** ページで、詳細を表示するイメージを選択します。

Amazon ECR プライベートリポジトリからローカル環境へのイメージのプル

Amazon ECR で利用可能な Docker イメージを実行する場合、docker pull コマンドを使用してローカル環境にプルします。これは、デフォルトのレジストリまたは別の AWS アカウントに関連付けられたレジストリから実行できます。

Amazon ECS タスク定義で Amazon ECR イメージを使用する場合は、「Amazon ECS での Amazon ECR イメージの使用」を参照してください。

> **Important**
>
> アーカイブされたイメージをプルすることはできません。アーカイブされたイメージは、プルする前に復元する必要があります。イメージのアーカイブと復元の詳細については、「」を参照してくださいAmazon ECR でのイメージのアーカイブ。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリへの認証を行って、Amazon ECR リポジトリに対するイメージのプッシュまたはプルを行う前に、IAM ポリシーを介して `ecr:GetAuthorizationToken` API への呼び出しを行う許可が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するための AWS 管理ポリシーがいくつか用意されています。Amazon ECR の AWS マネージドポリシーの詳細については、「」を参照してくださいAWS Amazon Elastic Container Registry の マネージドポリシー。

Amazon ECR リポジトリから Docker イメージをプルするには

1. イメージのプル元になる Amazon ECR レジストリに対して Docker クライアントを認証します。認証トークンは、使用するレジストリごとに取得する必要があり、トークンは 12 時間有効です。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。
2. (オプション) プルするイメージを識別します。
   - レジストリ内のリポジトリは、aws ecr describe-repositories コマンドを使用してリストできます。

```
aws ecr describe-repositories
```

上記のレジストリの例には、 amazonlinux というリポジトリがあります。

- リポジトリ内のイメージは、aws ecr describe-images コマンドで記述することができます。

```
aws ecr describe-images --repository-name amazonlinux
```

上記のリポジトリ例には、latest および 2016.09 というタグが付けられたイメージが、イメージダイジェスト sha256:f1d4ae3f7261a72e98c6ebefe9985cf10a0ea5bd762585a43e0700ed99863807 とともにあります。

3. docker pull コマンドを使用してイメージをプルします。イメージ名の形式は、タグを使用してプルする場合は *registry*/*repository* [:*tag*]、ダイジェストを使用してプルする場合は *registry* /*repository*[@*digest*] とします。

```
docker pull aws_account_id.dkr.ecr.us-west-2.amazonaws.com/amazonlinux:latest
```

> **Important**
>
> *repository-url* not found: does not exist or no pull access エラーが表示された場合は、Amazon ECR で Docker クライアントを認証する必要があります。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。

Amazon Linux コンテナイメージのプル

Amazon Linux コンテナイメージは、Amazon Linux AMI に含まれているのと同じソフトウェアコンポーネントから構築されます。Amazon Linux コンテナイメージは、Docker ワークロードのベースイメージとして任意の環境で使用できます。Amazon EC2 でアプリケーションに Amazon Linux AMI を使用する場合、Amazon Linux コンテナイメージを使用してアプリケーションをコンテナ化できます。

ローカル開発環境で Amazon Linux コンテナイメージを使用し、Amazon ECS AWS を使用してアプリケーションを にプッシュできます。詳細については、「Amazon ECS での Amazon ECR イメージの使用」を参照してください。

Amazon Linux コンテナイメージは、Amazon ECR Public および Docker Hub で入手できます。Amazon Linux コンテナイメージのサポートについては、AWS デベロッパーフォーラムにアクセスしてください。

Amazon ECR Public から Amazon Linux コンテナイメージをプルするには

1. Amazon Linux Public レジストリで Docker クライアントを認証します。認証トークンは 12 時間有効です。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。

   > **Note**
   >
   > ecr-public コマンドは、バージョン 1.18.1.187 以降の AWS CLI で使用できます。ただし、 AWS CLIの最新バージョンを使用することをお勧めします。詳細については、「AWS Command Line Interface ユーザーガイド」の「Installing the AWS Command Line Interface」を参照してください。

   ```
   aws ecr-public get-login-password --region us-east-1 | docker login --username AWS
    --password-stdin public.ecr.aws
   ```

   出力は次のとおりです。

   ```
   Login succeeded
   ```

2. docker pull コマンドを使用して Amazon Linux コンテナイメージをプルします。Amazon ECR Public Gallery で Amazon Linux コンテナイメージを表示するには、「Amazon ECR Public Gallery - amazonlinux」を参照してください。

   ```
   docker pull public.ecr.aws/amazonlinux/amazonlinux:latest
   ```

3. (オプション) コンテナをローカルに実行します。

   ```
   docker run -it public.ecr.aws/amazonlinux/amazonlinux /bin/bash
   ```

Docker Hub から Amazon Linux コンテナイメージをプルするには

1. docker pull コマンドを使用して Amazon Linux コンテナイメージをプルします。

```
docker pull amazonlinux
```

2. (オプション) コンテナをローカルに実行します。

```
docker run -it amazonlinux:latest /bin/bash
```

Amazon ECR でのイメージの削除

使用が終わったイメージはリポジトリから削除できます。リポジトリでの作業が終了したら、リポジトリ全体とその中のすべてのイメージを削除できます。詳細については、「Amazon ECR でのプライベートリポジトリの削除」を参照してください。

イメージを手動で削除する代わりに、リポジトリのライフサイクルポリシーを作成すると、リポジトリ内のイメージのライフサイクル管理をより詳細に制御できます。ライフサイクルポリシーを使用すると、このプロセスが自動化されます。詳細については、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。

> **Note**
>
> リポジトリ内でイメージが混在している (一部のイメージは Amazon ECR が OCI v1.1 をサポートする前にプッシュされた) 場合、一部の署名にはイメージインデックスやそれを指すマニフェストリストが含まれます。そのため、OCI v1.1 より前のイメージを削除する場合、アーティファクトを削除するには、イメージを参照するマニフェストリストを手動で削除する必要がある場合があります。

イメージを削除するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、削除するイメージを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、削除するイメージを含むリポジトリを選択します。

5. リポジトリ: ***repository_name*** ページで、削除するイメージの左側にあるボックスを選択し、削除を選択します。
6. [Delete image(s)] ダイアログボックスで、選択したイメージを削除することを確認し、[Delete] を選択します。

イメージを削除するには (AWS CLI)

1. リポジトリ内のイメージを一覧表示します。タグ付けされたイメージには、イメージダイジェストおよび関連するタグのリストの両方が含まれます。タグ付けされていないイメージには、イメージダイジェストのみが含まれます。

```
aws ecr list-images \
     --repository-name my-repo
```

2. (オプション) 削除するイメージに関連付けられたタグを指定して、イメージの不要なタグを削除します。イメージの最後のタグを削除すると、イメージが削除されます。

```
aws ecr batch-delete-image \
     --repository-name my-repo \
     --image-ids imageTag=tag1 imageTag=tag2
```

3. イメージダイジェストを指定して、タグ付けされたイメージ、またはタグ付けされていないイメージを削除します。ダイジェストを参照してイメージを削除する場合、イメージとそのすべてのタグは削除されます。

```
aws ecr batch-delete-image \
     --repository-name my-repo \
     --image-ids imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE
```

   複数のイメージを削除するには、リクエストで複数のイメージタグまたはイメージダイジェストを指定します。

```
aws ecr batch-delete-image \
     --repository-name my-repo \
     --image-ids imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE
 imageDigest=sha256:f5t0e245ssffc302b13e25962d8f7a0bd304EXAMPLE
```

Amazon ECR でのイメージのアーカイブ

ECR アーカイブストレージクラスとは

Amazon ECR アーカイブストレージクラスは、コンテナイメージの低コストで長期的なストレージを提供する新しいストレージクラスです。Amazon ECR には 2 つのストレージクラスがあります。

- ECR 標準ストレージクラス – 定期的にアクセスされるアクティブなイメージのデフォルトのストレージクラス。
- ECR アーカイブストレージクラス – アクセスはまれですが、コンプライアンスまたは長期参照のために保持する必要があるイメージの低コストストレージクラス。アーカイブストレージクラスは、長期イメージ保持のための標準ストレージクラスと比較して、大量のイメージのコスト削減を実現します。料金の詳細については、「Amazon ECR の料金」を参照してください。

イメージをアーカイブするには、2 つのオプションがあります。まず、以下に基づいてイメージを自動的にアーカイブするようにライフサイクルルールを設定できます。

- イメージがプッシュされてからの時間
- イメージが最後にプルされてからの時間
- リポジトリ内のイメージの数

指定した期間にアーカイブされたイメージを完全に削除するように設定することもできます。詳細については、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。

Amazon ECR コンソールまたは を使用してイメージをアーカイブすることもできます AWS CLI。詳細については、「イメージのアーカイブ」を参照してください。

アーカイブされたイメージを再度使用する必要がある場合は、ECR Standard ストレージクラスに復元できます。ECR は 20 分以内にイメージを復元します。復元されたイメージは、新しくプッシュされたイメージのように動作し、復元が完了するとすぐに使用できます。復元されたイメージには、スキャン、レプリケーション、リポジトリのライフサイクルポリシーが適用されます。詳細については、「イメージの復元」を参照してください。

イメージのアーカイブ

Amazon ECR コンソールを使用して手動でイメージをアーカイブすることも AWS CLI、ライフサイクルポリシーを使用して自動的にアーカイブすることもできます。イメージがアーカイブされている場合:

- イメージはアーカイブストレージクラスに移動されます。
- アーカイブされたイメージはプルできません。アーカイブされたイメージのプルリクエストは 404 エラーで失敗します。
- イメージはプルできませんが、 describe-images コマンドを使用して記述することも、 list-images コマンドを使用してリストすることもできます。イメージのステータスは として表示されますARCHIVED。
- アーカイブされたイメージの最小ストレージ期間は 90 日間です。アーカイブに 90 日未満のイメージを削除するライフサイクルポリシーを設定することはできません。アーカイブ期間が 90 日未満のイメージを削除する必要がある場合は、 batch-delete-image API を使用する必要がありますが、90 日間の最小ストレージ期間に対して課金されます。
- イメージは、リポジトリビューのアーカイブ済みイメージタブに表示されます (このタブは、少なくとも 1 つのイメージがリポジトリにアーカイブされている場合にのみ表示されます) 。
- イメージは、復元するイメージを手動で選択するか、リポジトリにイメージを再プッシュすることで、アクティブなイメージとして復元できます。
- アーカイブの時間などの基準でイメージを削除するライフサイクルポリシーがリポジトリにある場合、イメージは削除されます。

AWS マネジメントコンソール

イメージをアーカイブするには

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、アーカイブするイメージを含むリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、アーカイブするイメージを含むリポジトリを選択します。
5. アーカイブするイメージを選択します。イメージの詳細が表示されます。
6. イメージをアーカイブするには、アーカイブボタンを選択し、プロンプトが表示されたら確認を選択します。

7. これがリポジトリ内の最初のアーカイブイメージである場合、新しいアーカイブイメージタブに新しくアーカイブされたイメージが表示されます。アーカイブされたイメージが他にもある場合、このイメージはそのタブに追加されます。

AWS CLI

イメージをアーカイブするには

- update-image-storage-class コマンドを使用して、ストレージクラスを に更新してイメージをアーカイブしますARCHIVE。

```
aws ecr update-image-storage-class \
    --repository-name my-repository \
    --image-id
 imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE \
    --target-storage-class ARCHIVE
```

ライフサイクルポリシーを使用してイメージをアーカイブするには

- ライフサイクルポリシーを使用してリポジトリのアーカイブルールを設定し、イメージを自動的にアーカイブできます。ライフサイクルポリシーを使用すると、次のような基準に基づいてイメージを自動的にアーカイブできます。
  - イメージがプッシュされてからの時間
  - イメージが最後にプルされてからの時間
  - アクティブにしておくイメージの最大数

  指定した期間アーカイブされたイメージを完全に削除するようにライフサイクルポリシーを設定することもできます。アーカイブアクションを含むライフサイクルポリシーの詳細と例については、「」を参照してくださいAmazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化。

  > **Note**
  >
  > アーカイブされたイメージの最小ストレージ期間は 90 日間です。アーカイブに 90 日未満のイメージを削除するライフサイクルポリシーを設定することはできません。アーカ

イブ期間が 90 日未満のイメージを削除する必要がある場合は、 batch-delete-image API を使用する必要がありますが、90 日間の最小ストレージ期間に対して課金されます。

describe-images コマンドを使用してイメージを記述すると、アーカイブされたイメージの は `image-status`になりますARCHIVED。でイメージ`image-status`をフィルタリングして、アーカイブされたイメージのみまたはアクティブなイメージのみを表示できます。

イメージの復元

アーカイブされたイメージを復元すると、ECR Archive ストレージクラスから ECR Standard ストレージクラスに戻ります。復元されたイメージは、標準のストレージレートで課金されます。復元プロセスは、新しいイメージが作成されたときに発生する同様のアクションを実行します。

- 復元が完了すると、イメージをプルできるようになります。復元には通常最大 20 分かかりますが、完了が速くなる場合があります。
- プッシュ時のスキャンがリポジトリに対して有効になっている場合、復元されたイメージがスキャンされます。イメージがアーカイブされる前の以前のスキャン結果は使用できないことに注意してください。
- リポジトリにレプリケーションが設定されている場合、復元時にレプリケーションが有効になっていると、復元されたイメージがレプリケートされます。
- 復元されたイメージがアクティブなイメージリストに表示されます。

イメージの復元には通常最大 20 分かかりますが、完了が速くなる場合があります。復元プロセス中、イメージはアーカイブされた状態のままであり、復元が完了するまでプルできません。

AWS マネジメントコンソール

アーカイブされたイメージを復元するには

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、復元するアーカイブイメージを含むリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、アーカイブされたイメージを含むリポジトリを選択します。
5. アーカイブされたイメージタブを選択します。

6. 復元するアーカイブ済みイメージを選択します。
7. 復元 を選択し、復元アクションを確認します。
8. 復元が完了するまで待ちます。復元が完了すると、イメージがアクティブなイメージリストに表示されます。

AWS CLI

アーカイブされたイメージを復元するには

- update-image-storage-class コマンドを使用して、ストレージクラスを に更新してアーカイブされたイメージを復元しますSTANDARD。

```
aws ecr update-image-storage-class \
    --repository-name my-repository \
    --image-id
 imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE \
    --target-storage-class STANDARD
```

describe-images コマンドを使用してイメージを記述すると、復元されるイメージの は image-statusになりますACTIVATING。値image-statusでイメージをフィルタリングACTIVATINGして、現在復元中のイメージを表示できます。

アーカイブされたイメージを復元するもう 1 つの方法は、イメージをリポジトリに再プッシュすることです。現在アーカイブされているイメージをプッシュすると、そのイメージはすぐに復元され、アーカイブから削除されます。

Amazon ECR でのイメージの再タグ付け

Docker Image Manifest V2 Schema 2 のイメージでは、put-image コマンドの --image-tag オプションを使用して、既存のイメージにもう一度タグを付けることができます。Docker でイメージをプルまたはプッシュしなくても、もう一度タグを付けることができます。大きなイメージの場合、このプロセスにより、イメージにもう一度タグを付けるために必要なネットワーク帯域幅と時間がかなり節約されます。

イメージにもう一度タグを付けるには (AWS CLI)

を使用してイメージにタグを付け直すには AWS CLI

1. batch-get-image コマンドを使用して、イメージを再タグ付けしてファイルに書き込むためのイメージマニフェストを取得します。この例では、リポジトリ *amazonlinux* 内の *latest* タグ付きイメージのマニフェストが *MANIFEST* という名前の環境変数に書き込まれます。

```
MANIFEST=$(aws ecr batch-get-image --repository-name amazonlinux --image-ids
 imageTag=latest --output text --query 'images[].imageManifest')
```

2. put-image コマンドの `--image-tag` オプションを使用して、新しいタグでイメージマニフェストを Amazon ECR に配置します。この例では、イメージには *2017.03* というタグが付きます。

> **Note**
>
> お使いのバージョンの で `--image-tag`オプションを使用できない場合は AWS CLI、最新バージョンにアップグレードします。詳細については、「AWS Command Line Interface ユーザーガイド」の「Installing the AWS Command Line Interface」を参照してください。

```
aws ecr put-image --repository-name amazonlinux --image-tag 2017.03 --image-
manifest "$MANIFEST"
```

3. 新しいイメージタグがイメージにアタッチされていることを確認します。次の出力では、イメージに `latest` と `2017.03` のタグが付けられています。

```
aws ecr describe-images --repository-name amazonlinux
```

出力は次のとおりです。

```
{
    "imageDetails": [
        {
            "imageSizeInBytes": 98755613,
            "imageDigest":
 "sha256:8d00af8f076eb15a33019c2a3e7f1f655375681c4e5be157a26EXAMPLE",
```

```
            "imageTags": [
                "latest",
                "2017.03"
            ],
            "registryId": "aws_account_id",
            "repositoryName": "amazonlinux",
            "imagePushedAt": 1499287667.0
        }
    ]
}
```

イメージにもう一度タグを付けるには (AWS Tools for Windows PowerShell)

を使用してイメージにタグを付け直すには AWS Tools for Windows PowerShell

1. Get-ECRImageBatch cmdletを使用して、もう一度タグを付けるイメージの説明を取得し、環境変数にそれを書き込みます。この例では、リポジトリ *amazonlinux* に*###* タグが付いたイメージが環境変数 *$Image* に書き込まれます。

   > **Note**
   >
   > システムで Get-ECRImageBatch cmdletが使用できない場合は、「AWS Tools for PowerShell ユーザーガイド」の「Setting up the AWS Tools for Windows PowerShell」を参照してください。

   ```
   $Image = Get-ECRImageBatch -ImageId @{ imageTag="latest" } -
   RepositoryName amazonlinux
   ```

2. *$Manifest* 環境変数にイメージのマニフェストを書き込みます。

   ```
   $Manifest = $Image.Images[0].ImageManifest
   ```

3. Write-ECRImage cmdletの -ImageTag オプションを使用して、イメージマニフェストを新しいタグで Amazon ECR に配置します。この例では、イメージには *2017.09* というタグが付きます。

   ```
   Write-ECRImage -RepositoryName amazonlinux -ImageManifest $Manifest -
   ImageTag 2017.09
   ```

4. 新しいイメージタグがイメージにアタッチされていることを確認します。次の出力では、イメージに latest と 2017.09 のタグが付けられています。

```
Get-ECRImage -RepositoryName amazonlinux
```

出力は次のとおりです。

```
ImageDigest                                                             ImageTag
-----------                                                             --------
sha256:359b948ea8866817e94765822787cd482279eed0c17bc674a7707f4256d5d497 latest
sha256:359b948ea8866817e94765822787cd482279eed0c17bc674a7707f4256d5d497 2017.09
```

Amazon ECR でのイメージタグの上書きの防止

リポジトリでタグのイミュータビリティをオンにすることで、イメージタグの上書きを防止できます。タグのイミュータビリティをオンにした後、リポジトリに既に存在しているタグが付いたイメージをプッシュすると ImageTagAlreadyExistsException エラーが返されます。タグのイミュータビリティはすべてのタグに影響します。一部のタグをイミュータブルにし、その他のタグをイミュータブルにしないということはできません。

AWS マネジメントコンソール および AWS CLI ツールを使用して、新しいリポジトリまたは既存のリポジトリのイメージタグのミュータビリティを設定できます。コンソールの手順を使用してリポジトリを作成するには、「イメージを保存するための Amazon ECR プライベートリポジトリの作成」を参照してください。

イメージタグのミュータビリティの設定 (AWS マネジメントコンソール)

イメージタグのミュータビリティを設定する手順

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、編集するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ] の [リポジトリ] を選択します。

   リポジトリが表示されない場合は、プライベートレジストリを選択してメニューを展開し、リポジトリを選択します。
4. [プライベートリポジトリ] ページで、イメージタグのミュータビリティ設定を設定するリポジトリ名の前にあるラジオボタンを選択します。

5. [アクション] を選択し、[編集] の[リポジトリ] を選択します。
6. [Image tag immutability] で、このリポジトリのタグのミュータビリティの設定を選択します。
   - [ミュータブル] – イメージタグを上書きする場合は、このオプションを選択します。Amazon ECR がキャッシュされたイメージを更新できるように、プルスルーキャッシュアクションを使用するリポジトリに推奨されます。さらに、いくつかのミュータブルタグのタグ更新を無効にするには、[Mutable tag exclusion] テキストボックスにタグ名を入力します。複数の類似したタグが一致するワイルドカード (*) を使用できます。
   - [イミュータブル] – イメージタグが上書きされるのを防ぎ、既存のタグを含むイメージをプッシュするときに、リポジトリ内のすべてのタグと除外に適用する場合、このオプションを選択します。Amazon ECR は、既存のタグを使用してイメージをプッシュしようとすると `ImageTagAlreadyExistsException` を返します。さらに、いくつかのイミュータブルタグのタグ更新を有効にするには、[Immutable tag exclusion] テキストボックスにタグ名を入力します。複数の類似したタグが一致するワイルドカード (*) を使用できます。
7. [Image scan settings] (イメージスキャン設定) では、基本的なスキャンに対してはリポジトリレベルでのスキャン設定を指定できますが、プライベートレジストリレベルでスキャン設定を指定することがベストプラクティスとなります。プライベートレジストリでスキャン設定を指定することで、拡張スキャンまたはベーシックスキャンのいずれかを有効にすることができ、スキャンするリポジトリを指定するフィルターも定義できます。詳細については、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。
8. [Encryption settings] (暗号化設定) は表示専用フィールドであり、リポジトリの作成後にリポジトリの暗号化設定を変更することはできません。
9. [保存] を選択してリポジトリ設定を更新します。

イメージタグのミュータビリティの設定 (AWS CLI)

タグが変更不可に設定されたリポジトリを作成するには

次のいずれかのコマンドを使用して、タグが変更不可似せていされた新しいイメージリポジトリを作成します。

- イメージタグのミュータビリティのある create-repository (AWS CLI)

```
aws ecr create-repository --repository-name name --image-tag-mutability IMMUTABLE --
region us-east-2
```

- イメージタグのミュータビリティ除外フィルターのある create-repository (AWS CLI)

```
aws ecr create-repository --repository-name name --image-tag-
mutability IMMUTABLE_WITH_EXCLUSION --image-tag-mutability-exclusion-filters
 filterType=WILDCARD,filter=filter-text --region us-east-2
```

- イメージタグのミュータビリティのある New-ECRRepository (AWS Tools for Windows PowerShell)

```
New-ECRRepository -RepositoryName name -ImageTagMutability IMMUTABLE -Region us-
east-2 -Force
```

- イメージタグのミュータビリティ除外フィルターのある New-ECRRepository (AWS Tools for Windows PowerShell)

```
New-ECRRepository -RepositoryName name -ImageTagMutability IMMUTABLE_WITH_EXCLUSION
 -ImageTagMutabilityExclusionFilter @{FilterType=WILDCARD Filter=filter-text} -
Region us-east-2 -Force
```

リポジトリのイメージタグのミュータビリティ設定を更新する手順

次のいずれかのコマンドを使用して、既存のリポジトリのイメージタグの変更可能性を更新します。

- put-image-tag-mutability (AWS CLI) とイメージタグのミュータビリティ

```
aws ecr put-image-tag-mutability --repository-name name --image-tag-
mutability IMMUTABLE --region us-east-2
```

- put-image-tag-mutability (AWS CLI) とイメージタグのミュータビリティ除外フィルター

```
aws ecr put-image-tag-mutability --repository-name name --image-tag-
mutability IMMUTABLE_WITH_EXCLUSION --image-tag-mutability-exclusion-filters
 filterType=WILDCARD,filter=latest --region us-east-2
```

- イメージタグのミュータビリティを備えた Write-ECRImageTagMutability (AWS Tools for Windows PowerShell)

```
Write-ECRImageTagMutability -RepositoryName name -ImageTagMutability IMMUTABLE -
Region us-east-2 -Force
```

- イメージタグのミュータビリティ除外フィルターを使用した Write-ECRImageTagMutability (AWS Tools for Windows PowerShell)

```
Write-ECRImageTagMutability -RepositoryName name -
ImageTagMutability IMMUTABLE_WITH_EXCLUSION -ImageTagMutabilityExclusionFilter
 @{FilterType=WILDCARD Filter=latest}
```

Amazon ECR でのコンテナイメージマニフェスト形式のサポート

Amazon ECR は次のコンテナイメージマニフェスト形式をサポートします。

- Docker Image Manifest V2 Schema 1 (Docker バージョン 1.9 以前で使用)
- Docker Image Manifest V2 Schema 2 (Docker バージョン 1.10 以降で使用)
- Open Container Initiative (OCI) 仕様 (v1.0 および v1.1)

Docker Image Manifest V2 Schema 2 のサポートでは、次の機能が提供されます。

- 単一のイメージに複数のタグを使用する機能。
- Windows コンテナイメージを保存するためのサポート。

Amazon ECR イメージマニフェストの変換

Amazon ECR に対してイメージをプッシュまたはプルする場合、コンテナエンジンクライアント (Docker など) はレジストリと通信し、クライアントとレジストリでイメージに対して使用することが理解されているマニフェスト形式について合意します。

Docker バージョン 1.9 以前で Amazon ECR にイメージをプッシュする場合、イメージマニフェストは Docker Image Manifest V2 Schema 1 形式で保存されます。Docker バージョン 1.10 以降で Amazon ECR にイメージをプッシュする場合、イメージマニフェストは Docker Image Manifest V2 Schema 2 形式で保存されます。

Amazon ECR からタグでイメージをプルすると、Amazon ECR はリポジトリに格納されているイメージマニフェスト形式を返します。その形式が返されるのは、その形式がクライアントによって理解される場合のみです。保存されているイメージマニフェスト形式がクライアントによって理解されない場合、Amazon ECR はイメージマニフェストを理解される形式に変換します。たとえば、Docker 1.9 クライアントが Docker Image Manifest V2 Schema 2 で保存されているイメージマ

ニフェストをリクエストすると、Amazon ECR は Docker Image Manifest V2 Schema 1 形式でマニフェストを返します。次の表は、タグ別にイメージをプルしたときに、Amazon ECR でサポートされる利用可能な変換を示します。

クライアントによってリクエストされたスキーマ	V2、スキーマ 1 形式で ECR にプッシュされる	V2、スキーマ 2 形式で ECR にプッシュされる	OCI 形式で ECR にプッシュされる
V2、スキーマ 1	変換は必要ありません	V2、スキーマ 1 に変換される	変換は利用できません
V2、スキーマ 2	利用可能な変換はなく、クライアントは V2、スキーマ 1 にフォールバックする	変換は必要ありません	V2、スキーマ 2 に変換される
OCI	変換は利用できません	OCI に変換される	変換は必要ありません

> **Important**
>
> ダイジェスト別にイメージをプルした場合、利用可能な翻訳はありません。クライアントは、Amazon ECR に保存されているイメージマニフェストの形式を理解する必要があります。Docker 1.9 以前のクライアントで、Docker Image Manifest V2 Schema 2 イメージをダイジェストを使用してリクエストする場合、イメージのプルは失敗します。詳細については、Docker ドキュメントの「Registry compatibility」を参照してください。
> この例では、タグで同じイメージをリクエストした場合、Amazon ECR はクライアントが理解できる形式にイメージマニフェストを変換します。イメージのプルが成功します。

Amazon ECS での Amazon ECR イメージの使用

Amazon ECR プライベートリポジトリを使用して、Amazon ECS タスクがプルする可能性のあるコンテナイメージとアーティファクトをホストできます。これが機能するためには、Amazon ECS (または Fargate) コンテナエージェントに、`ecr:BatchGetImage`、`ecr:GetDownloadUrlForLayer`、および `ecr:GetAuthorizationToken` API を作成する許可が必要です。

必要な IAM 許可

次の表は、タスクが Amazon ECR プライベートリポジトリからプルするために必要な許可を提供する、起動タイプごとに使用する IAM ロールを示しています。Amazon ECS には、必要なアクセス許可を含むマネージド IAM ポリシーが用意されています。

起動タイプ	IAM ロール	AWS マネージド IAM ポリシー
Amazon EC2 インスタンスの Amazon ECS	Amazon ECS クラスターに登録されている Amazon EC2 インスタンスに関連付けられているコンテナインスタンスの IAM ロールを使用します。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「コンテナインスタンスの IAM ロール」を参照してください。	`AmazonEC2ContainerServiceforEC2Role` 詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「AmazonEC2ContainerServiceforEC2Role」を参照してください
Fargate の Amazon ECS	Amazon ECS タスク定義で参照するタスク実行 IAM ロールを使用します。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「タスク実行 IAM ロール」を参照してください。	`AmazonECSTaskExecutionRolePolicy` 詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「AmazonECSTaskExecutionRolePolicy」を参照してください。
外部インスタンスの Amazon ECS	Amazon ECS クラスターに登録されているオンプレミスサーバーまたは仮想マシン (VM) に関連付けられているコンテナインスタンスの IAM ロールを使用します。詳細については、「Amazon Elastic Container Service デベ	`AmazonEC2ContainerServiceforEC2Role` 詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「AmazonEC2Container

起動タイプ	IAM ロール	AWS マネージド IAM ポリシー
	ロッパーガイド」の「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。	ServiceforEC2Role」を参照してください。

Important

AWS マネージド IAM ポリシーには、使用に不要な追加のアクセス許可が含まれています。この場合、これらは Amazon ECR プライベートリポジトリからプルするために最低限必要なアクセス許可です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS タスク定義での Amazon ECR イメージの指定

Amazon ECS タスク定義を作成するときに、Amazon ECR プライベートリポジトリでホストされているコンテナイメージを指定できます。タスク定義では、Amazon ECR イメージ用に完全な `registry/repository:tag` の名前を使用するようにしてください。例えば、*aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-repository*:*latest*。

次のタスク定義スニペットは、Amazon ECS タスク定義の Amazon ECR でホストされるコンテナイメージを指定するために使用する構文を示しています。

```
{
    "family": "task-definition-name",
    ...
    "containerDefinitions": [
        {
            "name": "container-name",
            "image": "aws_account_id.dkr.ecr.region.amazonaws.com/my-
repository:latest",
            ...
        }
    ],
    ...
}
```

Amazon EKS での Amazon ECR イメージの使用

Amazon EKS で Amazon ECR イメージを使用できます。

Amazon ECR からイメージを参照する場合は、イメージに `registry/repository:tag` の形式の完全な名前を使用する必要があります 例えば、*aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-repository*:*latest*。

必要な IAM 許可

マネージドノード、セルフマネージドノード、または でホストされている Amazon EKS ワークロードがある場合は AWS Fargate、以下を確認してください。

- 管理ノードまたは自己管理ノードでホストされる Amazon EKS ワークロードの場合: Amazon EKS ワーカーノード IAM ロール (`NodeInstanceRole`) が必要です。Amazon EKS ワーカーノード IAM ロールには、Amazon ECR の次の IAM ポリシー許可が含まれている必要があります。

  JSON

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

```
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> `eksctl` または「Amazon EKS の開始方法」の CloudFormation テンプレートを使用してクラスターとワーカーノードグループを作成した場合、これらの IAM アクセス許可はデフォルトでワーカーノードの IAM ロールに適用されます。

- でホストされる Amazon EKS ワークロード AWS Fargate: Fargate ポッド実行ロールを使用します。これにより、プライベート Amazon ECR リポジトリからイメージをプルするアクセス許可がポッドに付与されます。詳細については、「Fargate ポッド実行ロールの作成」を参照してください。

Amazon EKS クラスターに Helm チャートをインストールする

Amazon ECR でホストされている Helm チャートを Amazon EKS クラスターにインストールできます。

前提条件

- Helm クライアントの最新バージョンをインストールします。これらのステップは、Helm バージョン `3.9.0` を使用して作成されました。詳細については、「Installing Helm」を参照してください。
- 少なくとも、 AWS CLI のバージョン `1.23.9` または `2.6.3` がコンピュータにインストールされています。詳細については、「Installing or updating the latest version of the AWS CLI」を参照してください。
- Helm チャートを Amazon ECR リポジトリにプッシュします。詳細については、「Amazon ECR プライベートリポジトリへの Helm チャートのプッシュ」を参照してください。

- Amazon EKS で使用する kubectl を設定します。詳細については、Amazon EKS ユーザーガイドの「Amazon EKS の kubeconfig を作成する」を参照してください。次のコマンドがクラスターに対して正常に実行された場合は、正しく設定されています。

```
kubectl get svc
```

Amazon EKS クラスターへの Helm チャートのインストール手順

1. Helm チャートがホストされている Amazon ECR レジストリで Helm クライアントを認証します。認証トークンは、使用するレジストリごとに取得する必要があり、トークンは 12 時間有効です。詳細については、「Amazon ECR でのプライベートレジストリ認証」を参照してください。

```
aws ecr get-login-password \
     --region us-west-2 | helm registry login \
     --username AWS \
     --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

2. グラフをインストールします。*helm-test-chart*をリポジトリに、*0.1.0*をHelm チャートのタグに置き換えます。

```
helm install ecr-chart-demo oci://aws_account_id.dkr.ecr.region.amazonaws.com/helm-test-chart --version 0.1.0
```

出力は次のようになります。

```
NAME: ecr-chart-demo
LAST DEPLOYED: Tue May 31 17:38:56 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

3. チャートのインストールを検証します。

```
helm list -n default
```

出力例:

```
NAME            NAMESPACE       REVISION        UPDATED
     STATUS          CHART                   APP VERSION
ecr-chart-demo  default         1               2022-06-01 15:56:40.128669157 +0000
 UTC deployed        helm-test-chart-0.1.0   1.16.0
```

4. (オプション) インストールされている Helm チャート ConfigMap を確認します。

```
kubectl describe configmap helm-test-chart-configmap
```

5. 終了したら、クラスターからチャートのリリースを削除できます。

```
helm uninstall ecr-chart-demo
```

Amazon ECR でイメージに署名する

Amazon ECR は と統合 AWS Signer され、マネージド署名 (自動、推奨) と手動署名 (クライアント側) の 2 つの方法でコンテナイメージに署名できます。コンテナイメージと署名の両方をプライベートのリポジトリに保存することができます。

署名方法を選択する

Amazon ECR は、コンテナイメージに署名するための 2 つの方法をサポートしています。

マネージド署名 (推奨)

マネージド署名は、イメージが Amazon ECR にプッシュされると、暗号化署名を自動的に生成します。この方法により、セットアップが簡素化されます。マネージド署名は、ほとんどのユーザーに推奨されるアプローチです。詳細については、「マネージド署名」を参照してください。

手動署名

手動署名では、Notation CLI と AWS Signer プラグインを使用してイメージに署名してから、Amazon ECR にプッシュします。この方法では、署名プロセスをより詳細に制御でき、プッシュワークフローの外部でイメージに署名する必要がある場合や、署名オペレーションをきめ細かく制御する必要がある場合に便利です。詳細については、「手動署名」を参照してください。

考慮事項

Amazon ECR イメージ署名を使用する場合は、次の点を考慮する必要があります。

- リポジトリに保存されている署名は、リポジトリあたりの最大イメージ数のサービスクォータカウントされます。各署名は、リポジトリクォータごとにイメージに対して 1 つのアーティファクトとしてカウントされます。詳細については、「Amazon ECR のサービスクォータ」を参照してください。
- リポジトリにリファレンスアーティファクトが存在する場合、Amazon ECR ライフサイクルポリシーは、サブジェクトイメージの削除から 24 時間以内にそれらのアーティファクトを自動的にクリーンアップします。

マネージド署名

Amazon ECR マネージド署名は、イメージが Amazon ECR にプッシュされたときに AWS Signer を使用して暗号化署名を生成することで、コンテナイメージを自動的に署名します。これにより、クライアント側のツールをインストールして設定する必要がなくなり、レジストリ設定として署名を一元管理できます。

前提条件

マネージド署名を設定するには、1 つ以上の Signer 署名プロファイルを参照する Amazon ECR を使用して署名設定を作成し、オプションで、イメージに署名するリポジトリを制限するリポジトリフィルターを作成します。設定すると、Amazon ECR マネージド署名は、イメージをプッシュするエンティティの ID を使用してプッシュされるときに、イメージに自動的に署名します。

マネージド署名を設定する前に、以下が必要です。

- 署名者署名プロファイル — 少なくとも 1 つの署名者署名プロファイルを作成します。署名プロファイルは、Amazon ECR で署名オペレーションを実行するために使用できる一意の AWS Signer リソースです。署名プロファイルを使用すると、コンテナイメージや AWS Lambda デプロイバンドルなどのコードアーティファクトに署名して検証できます。各署名プロファイルは、署名する署名プラットフォーム、プラットフォーム ID、およびその他のプラットフォーム固有の情報を指定します。たとえば、署名プロファイル ARN は次のようになります: `arn:partition:signer:region:account-id:/signing-profiles/profile-name`。
- IAM アクセス許可 — イメージをプッシュする IAM プリンシパルには、関連する Signer 署名プロファイルおよび関連する ECR リポジトリにアクセスするために必要な IAM アクセス許可が必要です。IAM プリンシパルのアイデンティティベースのポリシーを変更して、ECR リポジトリオペレーションと署名者署名オペレーションの両方のアクセス許可を含める必要があります。次のポリシー例は、必要なアクセス許可を示しています。

```
{
   "Version": "2012-10-17",
   "Statement": [
   {
       "Sid":"UploadSignaturePermissions",
       "Effect":"Allow",
       "Action":[
           "ecr:CompleteLayerUpload",
           "ecr:UploadLayerPart",
           "ecr:InitiateLayerUpload",
```

```
            "ecr:BatchCheckLayerAvailability",
            "ecr:PutImage"
        ],
        "Resource":"arn:aws:ecr:region:account-id:repository/repository-name"
    },
    {
        "Sid": "SignPermissions",
        "Effect": "Allow",
        "Action": [
            "signer:SignPayload"
        ],
        "Resource": "arn:aws:signer:region:account-id:/signing-profiles/signing-profile-
name"
    }
    ]
}
```

Amazon ECR マネージド署名を使用すると、複数の署名ルール (レジストリあたり最大 10 個) を作成して、より強力なセキュリティ境界を作成できます。たとえば、複数のビルドパイプラインを実行し、各パイプラインが署名できるリポジトリを制限できます。各ルール内で、署名プロファイルを設定し、リポジトリ名フィルターを指定します。新しいイメージがプッシュされると、Amazon ECR はイメージに署名できる署名ルールと署名プロファイルに一致します。複数の一致がある場合、Amazon ECR は複数の署名を生成します。

> **Note**
>
> 署名を手動で検証しても、Notation CLI をインストールする必要があります。

> **Note**
>
> Amazon ECR マネージド署名は、 AWS Signer を使用したコンテナイメージ署名が利用可能なすべての AWS リージョンで使用できます。

開始方法

マネージド署名を設定するには、次の手順に従います。署名者署名プロファイルへの参照を Amazon ECR に提供し、オプションで、イメージに署名するリポジトリを制限するフィルターを指定します。

AWS マネジメントコンソール

を使用してマネージド署名を設定するには、次の手順に従います AWS マネジメントコンソール。

1. Amazon ECR コンソールを開きます。左側のナビゲーションペインで、プライベートレジストリ、機能と設定、マネージド署名を選択します。
2. ルールの署名ページで、ルールの作成を選択します。
3. 署名プロファイルページの「署名者プロファイルの選択 AWS 」で、「新しい AWS 署名者プロファイルの作成」を選択し、プロファイル名を入力し、オプションで署名の有効期間を変更します。[次へ] を選択します。
4. 「フィルター」ページの「リポジトリの選択」の下に、リポジトリ名フィルターを入力します。[次へ] を選択します。
5. 確認と作成ページで、入力したAWS 署名者プロファイルとリポジトリ名フィルターを確認します。すべて正しい場合は、保存 を選択します。

AWS CLI

マネージド署名を設定するには、次の AWS CLI コマンドを使用します。

- 署名ルールを作成する

  署名プロファイル ARN を使用して署名設定を作成します。次の内容を含む JSON ファイルを作成します。

```
{
    "rules": [
        {
            "signingProfileArn": "arn:aws:signer:region:account-id:/signing-
profiles/profile-name",
            "repositoryFilters": [
                {
                    "filter": "test*",
                    "filterType": "WILDCARD_MATCH"
                }
            ]
        }
    ]
}
```

次に、以下のコマンドを実行します。

```
aws ecr --region region \
    put-signing-configuration \
    --signing-configuration file://signing-config.json
```

署名設定を含む API レスポンスが表示されます。

- 署名設定を表示する

  署名設定を取得します。

  ```
  aws ecr --region region \
      get-signing-configuration
  ```

  署名設定を含む API レスポンスが表示されます。

- イメージ署名ステータスを確認する

  イメージをリポジトリにプッシュします。例えば、次のようになります。

  ```
  docker pull ubuntu

  IMAGE_NAME="account-id.dkr.ecr.region.amazonaws.com/repository-name"
  IMAGE_TAG="${IMAGE_NAME}:test-1"

  docker tag ubuntu $IMAGE_TAG
  docker push $IMAGE_TAG
  ```

  プッシュしたら、イメージタグを使用して署名ステータスを確認します。

  ```
  aws ecr --region region \
      describe-image-signing-status \
      --repository-name repository-name \
      --image-id imageTag=test-1
  ```

  リポジトリ名が署名設定で定義されたリポジトリフィルターと一致する場合、API レスポンスに署名ステータスが表示されます。ステータスが成功すると、署名がリポジトリにプッシュされます。

- 署名設定を削除する

署名設定を削除します。

```
aws ecr --region region \
    delete-signing-configuration
```

削除された署名設定を含む API レスポンスが表示されます。

考慮事項

マネージド署名には、次の制限と機能が適用されます。

- クロスリージョン署名はサポートされていません — 署名プロファイルは Amazon ECR レジストリと同じリージョンにある必要があります。あるリージョンの署名プロファイルを使用して、別のリージョンにあるレジストリ内のイメージに署名することはできません。
- クロスアカウント署名がサポートされています — 署名プロファイルは、Amazon ECR レジストリとは異なるアカウントに配置できます。これにより、組織は署名プロファイルを一元管理しながら、他のアカウントの開発者が使用できるようになります。詳細については、「 AWS Signer デベロッパーガイド」の「署名者のクロスアカウント署名を設定する」を参照してください。
- 署名は署名できません — 署名自体に署名することはできません。署名できるのはコンテナイメージのみです。

署名の検証

コンテナイメージに署名したら、署名を検証して、イメージが改ざんされておらず、信頼できるソースからのものであることを確認できます。Amazon ECR は、署名を検証するためのいくつかの方法をサポートしています。

Amazon EKS によるマネージド検証

Amazon EKS は、署名の自動検証のためのネイティブ統合を提供します。Amazon EKS クラスターで署名検証を設定すると、コンテナの実行を許可する前に、サービスによってイメージ署名が自動的に検証されます。署名検証の設定の詳細については、「Amazon EKS ユーザーガイド」の「デプロイ中にコンテナイメージの署名を検証する」を参照してください。

Amazon ECS 用の Lambda アドミッションコントローラー

Amazon ECS には、サービスのデプロイ中にカスタムロジックを実行できるようにするサービスライフサイクルフックが用意されています。これらのフックは、デプロイプロセスの特定の時点で AWS Lambda 関数をトリガーできるため、サービスの開始を許可する前にコンテナイメージの署名を検証できます。詳細については、「 AWS Signer デベロッパーガイド」の「Amazon ECS のコンテナイメージ署名の検証」を参照してください。

Notation CLI による手動検証

Notation CLI を使用して署名を手動で検証できます。この方法では、ローカルマシンまたは検証環境に Notation CLI をインストールして設定する必要があります。Notation CLI を使用してイメージを検証する詳細な手順については、「 AWS Signer デベロッパーガイド」の「署名後にローカルでイメージを検証する」を参照してください。

Notation クライアントの認証を設定する

Notation CLI を使用して手動署名または署名を手動で検証する場合は、Amazon ECR に対して認証できるように Notation クライアントを設定する必要があります。Notation クライアントをインストールしたのと同じホストに Docker がインストールされている場合、Notation では Docker クライアントで使用されるのと同じ認証方法が再利用されます。Docker コマンド`login`と `logout` コマンドでは、Notation コマンド`sign`と `verify` コマンドが同じ認証情報を使用でき、Notation を個別に認証する必要はありません。Notation クライアントの認証設定の詳細については、Notary Project ドキュメントの「Authenticate with OCI-compliant registries」を参照してください。

Docker や Docker 認証情報を使用するその他のツールを使用していない場合は、認証情報ストアとして Amazon ECR Docker 認証情報ヘルパーを使用することをお勧めします。Amazon ECR 認証情報ヘルパーのインストールと設定方法の詳細については、「Amazon ECR Docker 認証情報ヘルパー」を参照してください。

手動署名

手動署名では、Notation CLI と AWS Signer プラグインを使用してイメージに署名してから、Amazon ECR にプッシュします。この方法では、署名プロセスをより詳細に制御でき、プッシュワークフローの外部でイメージに署名する必要がある場合や、署名オペレーションをきめ細かく制御する必要がある場合に便利です。

Notation CLI と を使用してコンテナイメージに署名する詳細な手順については AWS Signer、「AWS Signer デベロッパーガイド」の「Sign container images in Signer」および「関連トピック」を参照してください。

前提条件

開始するには、以下の前提条件を満たしておく必要があります。

- AWS CLIの最新バージョンがインストールされ、設定されている。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLIの最新バージョンのインストールまたは更新」を参照してください。
- Notation CLI と Notation 用 AWS Signer プラグインをインストールします。詳細については、「AWS Signer デベロッパーガイド」の「Prerequisites for signing container images」を参照してください。
- 署名するコンテナイメージを Amazon ECR プライベートリポジトリに保存します。詳細については、「Amazon ECR プライベートリポジトリへのイメージのプッシュ」を参照してください。

Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる

Amazon ECR イメージスキャンを使用すると、コンテナイメージ内のソフトウェア脆弱性を特定できます。次のスキャンタイプが提供されています。

> **Important**
>
> 拡張スキャンと基本スキャンを切り替えると、以前に確立されたスキャンが使用できなくなります。スキャンを再度設定する必要があります。ただし、以前のスキャンタイプに戻すと、確立されたスキャンが使用可能になります。

> **Note**
>
> アーカイブされたイメージはスキャンできません。アーカイブされたイメージは、スキャンする前に復元する必要があります。イメージのアーカイブと復元の詳細については、「」を参照してくださいAmazon ECR でのイメージのアーカイブ。

- 拡張スキャン – Amazon ECR は Amazon Inspector と統合され、リポジトリの自動継続的なスキャンを提供します。コンテナイメージは、オペレーティングシステムとプログラミング言語パッケージの両方の脆弱性についてスキャンされます。新しい脆弱性が発生すると、スキャン結果が更新され、Amazon Inspector は EventBridge にイベントを発行してユーザーに通知します。拡張スキャンでは、以下が提供されます。
  - OS およびプログラミング言語のパッケージに関する脆弱性。
  - 2 つのスキャン頻度: プッシュ時スキャンと継続的スキャン。
- 基本スキャン – Amazon ECR は、共通脆弱性識別子 (CVEs) データベースを備えた AWS ネイティブテクノロジーを使用して、オペレーティングシステムの脆弱性をスキャンします。

  ベーシックスキャンでは、プッシュ時にスキャンするようにリポジトリを設定します。手動スキャンを実行すると Amazon ECR によってスキャン結果のリストが提供されます。基本スキャンでは、以下が提供されます。
  - OS スキャン
  - 2 つのスキャン頻度: 手動スキャンとプッシュ時スキャン。

> **Important**
>
> 新しいバージョンの Amazon ECR Basic Scanning では、スキャン結果を返すために DescribeImages API レスポンスの imageScanFindingsSummary 属性と imageScanStatus 属性を使用しません。代わりに、 DescribeImageScanFindings API を使用してください。詳細については、「DescribeImageScanFindings」を参照してください。

Amazon ECR でスキャンするリポジトリを選択するためのフィルター

プライベートレジストリに対するイメージスキャンを設定する際には、フィルターを使用してスキャンするリポジトリを選択できます。

基本スキャンが使用されている場合は、プッシュフィルターでスキャンを指定して、新しいイメージがプッシュされたときにイメージスキャンを実行するように設定されるリポジトリを指定できます。プッシュ時の基本的なスキャンスキャンフィルターに一致しないリポジトリは、手動スキャン頻度に設定されます。つまり、スキャンを実行するには、スキャンを手動でトリガーする必要があります。

強化スキャンが使用されている場合は、プッシュ時のスキャンと連続スキャン用に別々のフィルターを指定できます。拡張スキャンフィルターに一致しないリポジトリでは、スキャンが無効になります。拡張スキャンを使用し、複数のフィルターが同じリポジトリに一致するプッシュ時スキャンと連続スキャンに別々のフィルターを指定すると、Amazon ECR はそのリポジトリのスキャンオンプッシュフィルターに対して連続スキャンフィルターを適用します。

フィルターのワイルドカード

フィルターを指定すると、ワイルドカードを含まないフィルターは、そのフィルターを含むすべてのリポジトリ名と一致します。ワイルドカード (*) を含むフィルターは、リポジトリ名のゼロ文字以上の文字をワイルドカードで置き換える任意のリポジトリ名と一致します。

次の表に、リポジトリ名を横軸に表し、フィルターの例を縦軸に指定する例を示します。

	prod	repo-prod	prod-repo	repo-prod-repo	prodrepo
prod	はい	はい	はい	はい	はい
*prod	はい	あり	なし	なし	いいえ
prod*	はい	なし	あり	なし	はい
prod	はい	はい	はい	はい	はい
prod*repo	いいえ	なし	あり	なし	はい

Amazon ECR で OS とプログラミング言語のパッケージの脆弱性を調べるためのイメージのスキャン

Amazon ECR 拡張スキャンは、コンテナイメージの脆弱性スキャンを提供する Amazon Inspector との統合です。コンテナイメージは、オペレーティングシステムとプログラミング言語パッケージの両方の脆弱性についてスキャンされます。スキャンの結果は、Amazon ECR と Amazon Inspector の両方で直接表示できます。Amazon Inspector の詳細については、Amazon Inspector ユーザーガイドの Scanning container images with Amazon Inspector を参照してください。

拡張スキャンでは、自動連続スキャン用に構成するリポジトリと、プッシュ時にスキャンするように構成するリポジトリを選択できます。これは、スキャンフィルターを設定することによって行われます。

拡張スキャンの考慮事項

Amazon ECR 拡張スキャンを有効にする前に、以下の点について考慮してください。

- この機能を使用するために Amazon ECR に追加料金はかかりませんが、イメージをスキャンするために Amazon Inspector の料金がかかります。この機能は、Amazon Inspector がサポートされているすべてのリージョンで利用できます。詳細については、以下を参照してください。
  - Amazon Inspector の料金 – Amazon Inspector の料金。
  - Amazon Inspector サポートされているリージョン – リージョンとエンドポイント。
- Amazon ECR 拡張スキャンは、Amazon EKS と Amazon ECS でイメージがどのように使用されるかを示します。イメージが最後に使用された日時を確認し、各イメージを使用するクラスターの

数を特定できます。この情報は、アクティブに使用されているイメージの脆弱性修復に優先順位を設定するのに役立ちます。新しく検出された脆弱性の影響を受ける可能性のあるクラスターをすばやく判断できます。これらの情報をリクエストしてレスポンスを表示する方法の詳細については、「DescribeImageScanFindings」を参照してください。

- Amazon Inspector では特定のオペレーティングシステムのスキャンがサポートされます。完全なリストについては、Amazon Inspector ユーザーガイドの「サポートされているオペレーティングシステム - Amazon ECR スキャン」を参照してください。
- Amazon Inspector は、サービスにリンクされた IAM ロールを使用します。このロールは、リポジトリに対する拡張スキャンを行うのに必要なアクセス許可を提供します。プライベートレジストリで拡張スキャンがオンになっている場合、サービスにリンクされた IAM ロールは Amazon Inspector によって自動的に作成されます。詳細については、「Amazon Inspector ユーザーガイド」の「Amazon Inspector でのサービスにリンクされたロールの使用」を参照してください。
- プライベートレジストリの拡張スキャンを最初に有効にすると、Amazon Inspector は、イメージプッシュタイムスタンプに基づいて、過去 14 日間に Amazon ECR にプッシュされたイメージのみを認識します。古い画像は `SCAN_ELIGIBILITY_EXPIRED` スキャンステータスになります。これらの画像を Amazon Inspector でスキャンしたい場合は、リポジトリに再度プッシュする必要があります。
- Amazon ECR プライベートレジストリで拡張スキャンがオンになっている場合、スキャンフィルターに一致するリポジトリは、拡張スキャンのみを使用してスキャンされます。フィルターと一致しないリポジトリのスキャン頻度は `Off` であり、スキャンされません。拡張スキャンを使用した手動スキャンはサポートされていません。詳細については、「Amazon ECR でスキャンするリポジトリを選択するためのフィルター」を参照してください。
- 複数のフィルターが同じリポジトリに一致するプッシュ時スキャンと連続スキャンに別々のフィルターを指定すると、Amazon ECR はそのリポジトリのスキャンオンプッシュフィルターに対して連続スキャンフィルターを適用します。
- 拡張スキャンがオンになっている場合、リポジトリのスキャン頻度が変更されると、Amazon ECR は EventBridge にイベントを送信します。Amazon Inspector は、初期スキャンが完了したとき、およびイメージスキャン結果が作成、更新、または閉じられたときに、EventBridge にイベントを発行します。

Amazon Inspector でのイメージの拡張スキャン期間の変更

拡張スキャンを有効にした後、Amazon ECR は、設定された期間、新しくプッシュされたイメージを連続してスキャンします。デフォルトでは、Amazon Inspector はイメージが削除されるか、拡張スキャンが無効になるまでリポジトリをモニタリングします。環境のニーズに合わせて、Amazon

Inspector コンソールでプッシュ日の期間 (最大有効期間) と再スキャン期間の両方を設定できます。リポジトリのスキャン期間が経過すると、スキャンステータスは SCAN_ELIGIBILITY_EXPIRED と表示されます。Amazon Inspector で Amazon ECR の再スキャン期間の設定の詳細については、「Amazon Inspector ユーザーガイド」の「Configuring the Amazon ECR re-scan duration」を参照してください。

Amazon ECR での拡張スキャンに必要な IAM アクセス許可

Amazon ECR 拡張スキャンには、Amazon Inspector サービスにリンクされた IAM ロールが必要です。また、拡張スキャンを有効化および使用する IAM プリンシパルには、スキャンで必要な Amazon Inspector API を呼び出すアクセス許可が必要です。プライベートレジストリで拡張スキャンがオンになっている場合、Amazon Inspector サービスにリンクされた IAM ロールは Amazon Inspector によって自動的に作成されます。詳細については、Amazon Inspector ユーザーガイドの Amazon Inspector でのサービスにリンクされたロールの使用 を参照してください。

次の IAM ポリシーは、拡張スキャンの有効化と使用に必要なアクセス許可を付与します。これには、Amazon Inspector がサービスにリンクされた IAM ロールを作成するために必要なアクセス許可、および拡張スキャンをオンまたはオフにして、およびスキャン結果の取得に必要な Amazon Inspector API のアクセス許可が含まれます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "inspector2:Enable",
                "inspector2:Disable",
                "inspector2:ListFindings",
                "inspector2:ListAccountPermissions",
                "inspector2:ListCoverage"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "inspector2.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Amazon ECR でのイメージの拡張スキャンの設定

プライベートレジストリに対する拡張スキャンをリージョンごとに設定します。

拡張スキャンを設定するための適切な IAM アクセス許可を持っていることを確認します。詳細については、「Amazon ECR での拡張スキャンに必要な IAM アクセス許可」を参照してください。

AWS マネジメントコンソール

プライベートレジストリに対する拡張スキャンをオンにする手順

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、スキャン設定を設定するリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ] を選択し、次に [設定] を選択します。
4. [Scanning configuration] (スキャン設定) ページの [Scan type] (スキャンタイプ) で、[Enhanced scanning] (拡張スキャン) を選択します。

   デフォルトでは、[拡張スキャン] を選択した場合、すべてのリポジトリが継続的にスキャンされます。
5. 特定のリポジトリを選択して継続的にスキャンするには、[すべてのリポジトリを継続的にスキャンする] ボックスをクリアして、フィルターを定義します。

   > **Important**
   >
   > ワイルドカードを含まないフィルターは、そのフィルターを含むすべてのリポジトリ名と一致します。ワイルドカード (*) を含むフィルターは、リポジトリ名のゼロ文字以上の文字がワイルドカードで置き換えられるリポジトリ名と一致します。フィル

ターの動作の例については、「the section called “フィルターのワイルドカード”」を参照してください。

a. リポジトリ名に基づいてフィルターを入力し、[フィルターの追加] を選択します。

b. イメージがプッシュされたときにスキャンするリポジトリを決定します。

- プッシュ時にすべてのリポジトリをスキャンするには、[プッシュ時にすべてのリポジトリをスキャンする] を選択します。
- 特定のリポジトリを選択してプッシュ時にスキャンするには、リポジトリ名に基づいてフィルターを入力し、[フィルターの追加] を選択します。

6. [保存] を選択します。

7. 拡張スキャンをオンにするリージョンごとに、これらの手順を繰り返します。

AWS CLI

次の AWS CLI コマンドを使用して、 を使用してプライベートレジストリの拡張スキャンを有効にします AWS CLI。スキャンフィルターは、 `rules` オブジェクトを使用して指定します。

- put-registry-scanning-configuration (AWS CLI)

  次の例では、プライベートレジストリの拡張スキャンをオンにします。デフォルトでは、`rules` が指定されていない場合、Amazon ECR はすべてのリポジトリのスキャン設定を継続スキャンに設定します。

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --region us-east-2
```

  次の例では、プライベートレジストリの拡張スキャンオンにして、スキャンフィルターを指定します。この例のスキャンフィルターは、名前に `prod` が含まれるすべてのリポジトリの継続スキャンをオンにします。

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --rules '[{"repositoryFilters" : [{"filter":"prod","filterType" :
 "WILDCARD"}],"scanFrequency" : "CONTINUOUS_SCAN"}]' \
```

```
    --region us-east-2
```

次の例では、プライベートレジストリの拡張スキャンをオンにして、複数のスキャンフィルターを指定します。この例のスキャンフィルターは、名前に prod が含まれるすべてのリポジトリの継続スキャンをオンにして、他のすべてのリポジトリに対してはプッシュ時のみスキャンをオンにします。

```
aws ecr put-registry-scanning-configuration \
    --scan-type ENHANCED \
    --rules '[{"repositoryFilters" : [{"filter":"prod","filterType" :
 "WILDCARD"}],"scanFrequency" : "CONTINUOUS_SCAN"},{"repositoryFilters" :
 [{"filter":"*","filterType" : "WILDCARD"}],"scanFrequency" : "SCAN_ON_PUSH"}]' \
    --region us-west-2
```

Amazon ECR の拡張スキャンについて送信される EventBridge イベント

拡張スキャンがオンになっている場合、リポジトリのスキャン頻度が変更されると、Amazon ECR は EventBridge にイベントを送信します。Amazon Inspector は、初期スキャンが完了したとき、およびイメージスキャン結果が作成、更新、または閉じられたときに、EventBridge にイベントを送信します。

リポジトリのスキャン頻度の変更に関するイベント

レジストリで拡張スキャンがオンになっている場合、拡張スキャンがオンになっているリソースで変更すると、Amazon ECR から次のイベントが送信されます。これには、新しいリポジトリの作成、リポジトリのスキャン頻度の変更、または拡張スキャンがオンになっているリポジトリでのイメージの作成または削除が含まれます。詳細については、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。

```
{
 "version": "0",
 "id": "0c18352a-a4d4-6853-ef53-0abEXAMPLE",
 "detail-type": "ECR Scan Resource Change",
 "source": "aws.ecr",
 "account": "123456789012",
 "time": "2021-10-14T20:53:46Z",
 "region": "us-east-1",
 "resources": [],
 "detail": {
  "action-type": "SCAN_FREQUENCY_CHANGE",
```

```
  "repositories": [{
    "repository-name": "repository-1",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-1",
    "scan-frequency": "SCAN_ON_PUSH",
    "previous-scan-frequency": "MANUAL"
   },
   {
    "repository-name": "repository-2",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-2",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   },
   {
    "repository-name": "repository-3",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-3",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   }
  ],
  "resource-type": "REPOSITORY",
  "scan-type": "ENHANCED"
 }
}
```

初期イメージスキャンのイベント (拡張スキャン)

レジストリで拡張スキャンがオンになっている場合、初期イメージスキャンが完了すると、Amazon Inspector によって以下のイベントが送信されます。`finding-severity-counts` パラメータは、重要度レベルが存在する場合にのみ、その値を返します。たとえば、イメージに `CRITICAL` レベルの結果が含まれていない場合、重要度のカウントは返されません。詳細については、「Amazon ECR で OS とプログラミング言語のパッケージの脆弱性を調べるためのイメージのスキャン」を参照してください。

イベントパターン:

```
{
  "source": ["aws.inspector2"],
  "detail-type": ["Inspector2 Scan"]
}
```

出力例:

```
{
    "version": "0",
    "id": "739c0d3c-4f02-85c7-5a88-94a9EXAMPLE",
    "detail-type": "Inspector2 Scan",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2021-12-03T18:03:16Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-sample"
    ],
    "detail": {
        "scan-status": "INITIAL_SCAN_COMPLETE",
        "repository-name": "arn:aws:ecr:us-east-2:123456789012:repository/amazon/
amazon-ecs-sample",
        "finding-severity-counts": {
            "CRITICAL": 7,
            "HIGH": 61,
            "MEDIUM": 62,
            "TOTAL": 158
        },
        "image-digest":
 "sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77e5EXAMPLE",
        "image-tags": [
            "latest"
        ]
    }
}
```

イメージスキャン結果の更新のイベント (拡張スキャン)

レジストリで拡張スキャンがオンになっている場合、イメージスキャン結果が作成、更新、または閉じられると、Amazon Inspector によって以下のイベントが送信されます。詳細については、「Amazon ECR で OS とプログラミング言語のパッケージの脆弱性を調べるためのイメージのスキャン」を参照してください。

イベントパターン:

```
{
  "source": ["aws.inspector2"],
  "detail-type": ["Inspector2 Finding"]
}
```

出力例:

```
{
    "version": "0",
    "id": "42dbea55-45ad-b2b4-87a8-afaEXAMPLE",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2021-12-03T18:02:30Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-sample/
sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77eEXAMPLE"
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "In libssh2 v1.9.0 and earlier versions, the SSH_MSG_DISCONNECT
 logic in packet.c has an integer overflow in a bounds check, enabling an attacker to
 specify an arbitrary (out-of-bounds) offset for a subsequent memory read. A crafted
 SSH server may be able to disclose sensitive information or cause a denial of service
 condition on the client system when a user connects to the server.",
        "findingArn": "arn:aws:inspector2:us-east-2:123456789012:finding/
be674aaddd0f75ac632055EXAMPLE",
        "firstObservedAt": "Dec 3, 2021, 6:02:30 PM",
        "inspectorScore": 6.5,
        "inspectorScoreDetails": {
            "adjustedCvss": {
                "adjustments": [],
                "cvssSource": "REDHAT_CVE",
                "score": 6.5,
                "scoreSource": "REDHAT_CVE",
                "scoringVector": "CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N",
                "version": "3.0"
            }
        },
        "lastObservedAt": "Dec 3, 2021, 6:02:30 PM",
        "packageVulnerabilityDetails": {
            "cvss": [
                {
                    "baseScore": 6.5,
                    "scoringVector": "CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N",
                    "source": "REDHAT_CVE",
                    "version": "3.0"
                },
```

```
                {
                    "baseScore": 5.8,
                    "scoringVector": "AV:N/AC:M/Au:N/C:P/I:N/A:P",
                    "source": "NVD",
                    "version": "2.0"
                },
                {
                    "baseScore": 8.1,
                    "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:H",
                    "source": "NVD",
                    "version": "3.1"
                }
            ],
            "referenceUrls": [
                "https://access.redhat.com/errata/RHSA-2020:3915"
            ],
            "source": "REDHAT_CVE",
            "sourceUrl": "https://access.redhat.com/security/cve/CVE-2019-17498",
            "vendorCreatedAt": "Oct 16, 2019, 12:00:00 AM",
            "vendorSeverity": "Moderate",
            "vulnerabilityId": "CVE-2019-17498",
            "vulnerablePackages": [
                {
                    "arch": "X86_64",
                    "epoch": 0,
                    "name": "libssh2",
                    "packageManager": "OS",
                    "release": "12.amzn2.2",
                    "sourceLayerHash":
"sha256:72d97abdfae3b3c933ff41e39779cc72853d7bd9dc1e4800c5294dEXAMPLE",
                    "version": "1.4.3"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "Update all packages in the vulnerable packages section to
their latest versions."
            }
        },
        "resources": [
            {
                "details": {
                    "awsEcrContainerImage": {
```

```
                        "architecture": "amd64",
                        "imageHash":
 "sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77e5EXAMPLE",
                        "imageTags": [
                            "latest"
                        ],
                        "platform": "AMAZON_LINUX_2",
                        "pushedAt": "Dec 3, 2021, 6:02:13 PM",
                        "lastInUseAt": "Dec 3, 2021, 6:02:13 PM",
                        "inUseCount": 1,
                        "registry": "123456789012",
                        "repositoryName": "amazon/amazon-ecs-sample"
                    }
                },
                "id": "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-
sample/sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77EXAMPLE",
                "partition": "N/A",
                "region": "N/A",
                "type": "AWS_ECR_CONTAINER_IMAGE"
            }
        ],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "CVE-2019-17498 - libssh2",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Dec 3, 2021, 6:02:30 PM"
    }
}
```

Amazon ECR での拡張スキャンの結果の取得

最後に完了した拡張イメージスキャンのスキャン結果を取得し、その結果を Amazon Inspector で開いて詳細を確認できます。検出されたソフトウェア脆弱性が、共通脆弱性識別子 (CVE) データベースに基づく重大度別にリストされます。

イメージをスキャンする際の一般的な問題のトラブルシューティングの詳細については、「Amazon ECR でのイメージスキャンのトラブルシューティング」を参照してください。

AWS マネジメントコンソール

AWS マネジメントコンソールを使用してイメージスキャンの結果を取得する手順は、次のとおりです。

イメージスキャンの結果を取得する手順

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、リポジトリがあるリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、スキャン結果を取得するイメージを含むリポジトリを選択します。
5. [イメージ] ページの [イメージタグ] 列で、スキャン結果を取得する対象のイメージタグを選択します。
6. Amazon Inspector コンソールで詳細を表示するには、[名前] 列で脆弱性の名前を選択します。

AWS CLI

次の AWS CLI コマンドを使用して、 を使用してイメージスキャンの結果を取得します AWS CLI。`imageTag` または `imageDigest` を使用してイメージを指定できます。どちらのイメージも list-images CLI コマンドを使用して取得できます。

- describe-image-scan-findings (AWS CLI)

  次の例では、イメージタグを使用しています。

  ```
  aws ecr describe-image-scan-findings \
       --repository-name name \
       --image-id imageTag=tag_name \
       --region us-east-2
  ```

  次の例では、イメージダイジェストを使用しています。

  ```
  aws ecr describe-image-scan-findings \
       --repository-name name \
       --image-id imageDigest=sha256_hash \
       --region us-east-2
  ```

Amazon ECR でイメージをスキャンして OS 脆弱性がないか調べる

Amazon ECR 基本スキャンでは、 AWS ネイティブテクノロジーを使用してコンテナイメージをスキャンし、ソフトウェアの脆弱性を調べます。ベーシックスキャンは、一般的なオペレーティングシステムの幅広いセットで脆弱性を検出し、50 を超えるデータフィードを調達して、一般的な脆弱性と露出 (CVEs) の検出結果を生成します。これらのソースには、ベンダーのセキュリティアドバイザリ、データフィード、脅威インテリジェンスフィード、国家脆弱性データベース (NVD)、MITRE などがあります。

Amazon ECR 基本スキャンは、「リージョンAWS 別のサービス」に記載されているすべてのリージョンでサポートされています。

Amazon ECR では、アップストリームのディストリビューションソースからの CVE の重要度が使用されます (使用可能な場合)。それ以外の場合は、共通脆弱性評価システム (CVSS) のスコアが使用されます。CVSS スコアは、NVD 脆弱性の重大度評価を取得するために使用できます。詳細については、「NVD 脆弱性の重大度」を参照してください。

Amazon ECR 基本スキャンでは、プッシュ時にスキャンするリポジトリを指定するフィルターがサポートされています。プッシュ時のスキャンフィルターと一致しないリポジトリは、手動スキャン頻度に設定されます。つまり、スキャンを手動で開始する必要があります。イメージは 24 時間ごとに 1 回スキャンできます。24 時間には、最初のプッシュ時スキャン (設定されている場合) とすべての手動スキャンが含まれます。基本スキャンでは、特定のレジストリで 24 時間あたり最大 100,000 個のイメージをスキャンできます。

最後に完了したイメージスキャンの結果は、各イメージに対して取得できます。イメージスキャンが完了すると、Amazon ECR は Amazon EventBridge にイベントを送信します。詳細については、「Amazon ECR イベントと EventBridge」を参照してください。

基本スキャンのオペレーティングシステムのサポート

セキュリティのベストプラクティスとして、また継続的なカバレッジのために、サポートされているバージョンのオペレーティングシステムを使い続けることをお勧めします。ベンダーのポリシーに従い、サポートが終了したオペレーティングシステムはパッチによる更新が行われなくなり、多くの場合、新しいセキュリティアドバイザリもリリースされなくなります。さらに、影響を受けるオペレーティングシステムが標準サポートが終了すると、既存のセキュリティアドバイザリと検出情報をフィードから削除するベンダーもあります。ディストリビューションに対するベンダーのサポートがなくなると、Amazon ECR は脆弱性のスキャンをサポートしなくなる可能性があります。サポート

が終了したオペレーティングシステムに対して Amazon ECR が生成した結果については、情報提供の目的でのみ使用してください。サポートされているオペレーティングシステムとバージョンの完全なリストについては、Amazon Inspector ユーザーガイド」の「サポートされているオペレーティングシステム - Amazon Inspector スキャン」を参照してください。 Amazon Inspector

Amazon ECR でのイメージの基本スキャンの設定

Amazon ECR では、デフォルトですべてのプライベートレジストリに対する基本スキャンがオンになっています。そのため、プライベートレジストリのスキャン設定を変更していなければ、基本スキャンをオンにする必要はありません。

次の手順を使用して、1 つ以上のプッシュ時スキャンフィルターを定義できます。

プライベートレジストリの基本スキャンをオンにする手順

1. https://console.aws.amazon.com/ecr/private-registry/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、スキャン設定を設定するリージョンを選択します。
3. ナビゲーションペインで、プライベートレジストリ、スキャンを選択します。
4. [Scanning configuration] (スキャン設定) ページの [Scan type] (スキャンタイプ) で [Basic scanning] (ベーシックスキャン) を選択します。
5. デフォルトでは、すべてのリポジトリは [Manual] (手動) スキャンに設定されます。オプションで、[プッシュ時にスキャンするフィルター] を指定して、プッシュ時のスキャンを設定できます。すべてのリポジトリまたは個々のリポジトリに対して、プッシュ時のスキャンを設定できます。詳細については、「Amazon ECR でスキャンするリポジトリを選択するためのフィルター」を参照してください。

> **Note**
>
> プッシュ時のスキャンがリポジトリに対して有効になっている場合、スキャンはアーカイブ後に復元されたイメージに対しても実行されます。復元されたイメージから古いスキャンは利用できません。

Amazon ECR で OS 脆弱性がないか調べるためのイメージの手動スキャン

リポジトリがプッシュ時スキャンを行うように設定されていない場合は、イメージスキャンを手動で開始できます。イメージは 24 時間ごとに 1 回スキャンできます。24 時間には、最初のプッシュ時スキャン (設定されている場合) とすべての手動スキャンが含まれます。

イメージをスキャンする際の一般的な問題のトラブルシューティングの詳細については、「Amazon ECR でのイメージスキャンのトラブルシューティング」を参照してください。

AWS マネジメントコンソール

AWS マネジメントコンソールを使用して手動イメージスキャンを開始するには、次の手順を実行します。

1. https://console.aws.amazon.com/ecr/private-registry/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、リポジトリを作成するリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、スキャンするイメージを含むリポジトリを選択します。
5. イメージページで、スキャンするイメージを選択し、[スキャン] を選択します。

AWS CLI

- start-image-scan (AWS CLI)

  次の例では、イメージタグを使用しています。

  ```
  aws ecr start-image-scan --repository-name name --image-id imageTag=tag_name --
  region us-east-2
  ```

  次の例では、イメージダイジェストを使用しています。

  ```
  aws ecr start-image-scan --repository-name name --image-id imageDigest=sha256_hash
   --region us-east-2
  ```

AWS Tools for Windows PowerShell

- Get-ECRImageScanFinding (AWS Tools for Windows PowerShell)

次の例では、イメージタグを使用しています。

```
Start-ECRImageScan -RepositoryName name -ImageId_ImageTag tag_name -Region us-
east-2 -Force
```

次の例では、イメージダイジェストを使用しています。

```
Start-ECRImageScan -RepositoryName name -ImageId_ImageDigest sha256_hash -
Region us-east-2 -Force
```

Amazon ECR での基本スキャンの結果の取得

最後に完了した基本イメージスキャンのスキャン結果を取得できます。検出されたソフトウェア脆弱性が、共通脆弱性識別子 (CVE) データベースに基づく重大度別にリストされます。

イメージをスキャンする際の一般的な問題のトラブルシューティングの詳細については、「Amazon ECR でのイメージスキャンのトラブルシューティング」を参照してください。

AWS マネジメントコンソール

AWS マネジメントコンソールを使用してイメージスキャンの結果を取得する手順は、次のとおりです。

イメージスキャンの結果を取得する手順

1. https://console.aws.amazon.com/ecr/private-registry/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、リポジトリを作成するリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. リポジトリページで、スキャン結果を取得するイメージを含むリポジトリを選択します。
5. [イメージ] ページの [イメージタグ] 列で、スキャン結果を取得する対象のイメージタグを選択します。

AWS CLI

次の AWS CLI コマンドを使用して、 を使用してイメージスキャンの結果を取得します AWS CLI。imageTag または　imageDigest を使用してイメージを指定できます。どちらのイメージも list-images CLI コマンドを使用して取得できます。

- describe-image-scan-findings (AWS CLI)

  次の例では、イメージタグを使用しています。

  ```
  aws ecr describe-image-scan-findings --repository-name name --image-id
   imageTag=tag_name --region us-east-2
  ```

  次の例では、イメージダイジェストを使用しています。

  ```
  aws ecr describe-image-scan-findings --repository-name name --image-id
   imageDigest=sha256_hash --region us-east-2
  ```

AWS Tools for Windows PowerShell

- Get-ECRImageScanFinding (AWS Tools for Windows PowerShell)

  次の例では、イメージタグを使用しています。

  ```
  Get-ECRImageScanFinding -RepositoryName name -ImageId_ImageTag tag_name -
  Region us-east-2
  ```

  次の例では、イメージダイジェストを使用しています。

  ```
  Get-ECRImageScanFinding -RepositoryName name -ImageId_ImageDigest sha256_hash -
  Region us-east-2
  ```

Amazon ECR でのイメージスキャンのトラブルシューティング

以下は、一般的なイメージスキャンの失敗です。このようなエラーは、Amazon ECR コンソールでイメージの詳細を表示するか、 API または　DescribeImageScanFindings API AWS CLI を使用して表示できます。

UnsupportedImageError

Amazon ECR でのベーシックイメージスキャニングがサポートされていないオペレーティングシステムを使用して構築されたイメージに対してベーシックスキャンを実行しようとすると、UnsupportedImageError エラーが発生することがあります。Amazon ECR は、Amazon Linux、Amazon Linux 2、Debian、Ubuntu、CentOS、Oracle Linux、Alpine、RHEL Linux ディストリビューションのメジャーバージョンのパッケージ脆弱性スキャニングをサポートします。ディストリビューションでベンダーからのサポートがなくなると、Amazon ECR は脆弱性のスキャンをサポートしなくなる可能性があります。Amazon ECRは、Docker スクラッチイメージから構築されたイメージのスキャンをサポートしません。

> **Important**
>
> 拡張スキャンを使用する場合、Amazon Inspector では特定のオペレーティングシステムおよびメディアタイプのスキャンがサポートされます。完全なリストについては、「Amazon Inspector ユーザーガイド」の「サポートされているオペレーティングとメディアタイプ」を参照してください。

UNDEFINED 深刻度が返される

深刻度が UNDEFINED のスキャン結果が返される場合があります。この問題の一般的な原因は以下のとおりです。

- この脆弱性に、CVE ソースによって優先度が割り当てられていなかった。
- この脆弱性に、Amazon ECR が認識しない優先度が割り当てられていた。

脆弱性の深刻度と説明を判断するには、ソースから直接 CVE を表示できます。

スキャンステータス SCAN_ELIGIBILITY_EXPIRED を理解する

プライベートレジストリに対して Amazon Inspector を使用した拡張スキャンが有効になっている場合にスキャンの脆弱性を表示すると、 SCAN_ELIGIBILITY_EXPIRED のスキャンステータスが表示されることがあります。この問題の最も一般的な原因は以下のとおりです。

- プライベートレジストリの拡張スキャンを最初にオンにすると、Amazon Inspector は、画像プッシュのタイムスタンプに基づいて、過去 30 日以内に Amazon ECR にプッシュされた画像のみを認識します。古い画像は SCAN_ELIGIBILITY_EXPIRED スキャンステータスになります。これ

らの画像を Amazon Inspector でスキャンしたい場合は、リポジトリに再度プッシュする必要があります。

- Amazon Inspector コンソールで ECR 再スキャン時間を変更し、その時間が経過すると、イメージのスキャンステータスは `expired` の理由コードで `inactive` に変更され、イメージに関するすべての結果はクローズされるようにスケジュールされます。その結果、Amazon ECR コンソールにスキャンステータスが `SCAN_ELIGIBILITY_EXPIRED` として一覧表示されます。

アップストリームレジストリと Amazon ECR プライベートレジストリの同期

プルスルーキャッシュルールを使用すると、アップストリームレジストリのコンテンツを Amazon ECR プライベートレジストリと同期できます。

Amazon ECR は現在、次のアップストリームレジストリのプルスルーキャッシュルールの作成をサポートしています。

- Amazon ECR パブリック、Kubernetes コンテナイメージレジストリ、Quay (認証は不要)
- Docker Hub、Microsoft Azure Container Registry、GitHub Container Registry、GitLab Container Registry、Chainguard Registry (シークレットによる AWS Secrets Manager 認証が必要)
- Amazon ECR (IAM AWS ロールによる認証が必要)

GitLab コンテナレジストリの場合、Amazon ECR は GitLab の Software as a Service (SaaS) オファリングでのみプルスルーキャッシュをサポートします。GitLab の SaaS オファリングの使用の詳細については、GitLab.comを参照してください。

シークレット (Docker Hub など) による認証を必要とするアップストリームレジストリの場合、認証情報を AWS Secrets Manager シークレットに保存する必要があります。Amazon ECR コンソールを使用して、認証されたアップストリームレジストリごとに Secrets Manager シークレットを作成できます。Secrets Manager コンソールを使用して Secrets Manager シークレットを作成する方法の詳細については、「シー AWS Secrets Manager クレットへのアップストリームリポジトリ認証情報の保存」を参照してください。

Amazon ECR の場合、アップストリームとダウンストリームの Amazon ECR レジストリが異なる AWS アカウントに属している場合、IAM ロールを作成する必要があります。IAM ロールの作成の詳細については 「クロスアカウント ECR から ECR へのプルスルーキャッシュに必要な IAM ポリシー」 をご参照ください。

アップストリームレジストリのプルスルーキャッシュルールを作成したら、Amazon ECR プライベートレジストリ URI を使用して、そのアップストリームレジストリからイメージをプルします。次に Amazon ECR はリポジトリを作成し、そのイメージをプライベートレジストリにキャッシュします。特定のタグのあるキャッシュされたイメージの後続のプルリクエストでは、Amazon ECR はアップストリームレジストリをチェックして、その特定のタグを持つイメージの新しいバージョンがあるかどうかを確認し、プライベートレジストリ内のイメージの更新を少なくとも 24 時間に 1 回試みます。

リポジトリ作成テンプレート

Amazon ECR でリポジトリ作成テンプレートのサポートが追加されました。これにより、Amazon ECR がプルスルーキャッシュルールを使用してユーザーの代わりに作成する新しいリポジトリの初期設定を、指定できるようになります。各テンプレートには、新しいリポジトリを特定のテンプレートと一致させるために使用されるリポジトリ名前空間プレフィックスが含まれています。テンプレートでは、リソースベースのアクセスポリシー、タグのイミュータビリティ、暗号化、ライフサイクルポリシーなど、すべてのリポジトリ設定の設定を指定できます。リポジトリ作成テンプレートの設定はリポジトリの作成時にのみ適用され、既存のリポジトリや他の方法で作成されたリポジトリには影響しません。詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。

プルスルーキャッシュルールを使用するための考慮事項

Amazon ECR のプルスルーキャッシュルールを使用する際には、以下の点について考慮する必要があります。

- 次のリージョンでは、プルスルーキャッシュルールの作成はサポートされていません。
  - 中国 (北京) (cn-north-1)
  - 中国 (寧夏) (cn-northwest-1)
  - AWS GovCloud (米国東部) (us-gov-east-1)
  - AWS GovCloud (米国西部) (us-gov-west-1)
- AWS Lambda は、プルスルーキャッシュルールを使用した Amazon ECR からのコンテナイメージのプルをサポートしていません。
- プルスルーキャッシュを使用してイメージをプルする場合、イメージを最初にプルするときに Amazon ECR FIPS サービスエンドポイントはサポートされません。ただし、Amazon ECR FIPS サービスエンドポイントは、その後のプルでも使用できます。
- 認証を必要とするアップストリームリポジトリの場合、イメージが Amazon ECR プライベートレジストリ URI を初めてプルするか、キャッシュを更新すると、イメージのプルはプルスルーキャッシュルールで設定された認証情報に関連付けられたユーザーによって開始されます。それ以降のプルは、顧客のプライベートレジストリのキャッシュから直接イメージを返します。
- 認証を必要としないアップストリームリポジトリの場合、イメージが Amazon ECR プライベートレジストリ URI を介してプルされると、イメージのプルは AWS IP アドレスによって開始されます。

- キャッシュされたイメージが Amazon ECR プライベートレジストリ URI を介してプルされると、Amazon ECR はアップストリームリポジトリを少なくとも 24 時間に 1 回チェックして、キャッシュされたイメージが最新バージョンであるかどうかを確認します。アップストリームレジストリに新しいイメージがある場合、Amazon ECR はキャッシュされたイメージの更新を試みます。このタイマーは、キャッシュされたイメージの最後のプルに基づいています。
- Amazon ECR が何らかの理由でアップストリームレジストリからイメージを更新できず、イメージがプルされた場合でも、最後にキャッシュされたイメージがプルされます。
- アップストリームのレジストリ認証情報を含む Secrets Manager シークレットを作成する場合、シークレット名には `ecr-pullthroughcache/` プレフィックスを使用する必要があります。シークレットは、プルスルーキャッシュルールが作成されたのと同じアカウントとリージョンにある必要もあります。
- プルスルーキャッシュルールを使用してマルチアーキテクチャイメージをプルすると、マニフェストリストとマニフェストリストで参照されている各イメージが Amazon ECR リポジトリにプルされます。特定のアーキテクチャのみをプルする場合は、マニフェストリストに関連付けられたタグではなく、アーキテクチャに関連付けられたイメージダイジェストまたはタグを使用してイメージをプルできます。
- Amazon ECR は、サービスにリンクされた IAM ロールを使用します。このロールは、Amazon ECR がユーザーに代わってキャッシュされたイメージのリポジトリを作成し、認証のために Secret Manager シークレット値を取得し、キャッシュされたイメージをプッシュするために必要なアクセス許可を提供します。プルスルーキャッシュルールを作成すると、サービスにリンクされた IAM ロールが自動的に作成されます。詳細については、「プルスルーキャッシュの Amazon ECR サービスにリンクされたロール」を参照してください。
- デフォルトで、キャッシュされたイメージをプルする IAM プリンシパルには、IAM ポリシーによってアクセス許可が付与されています。Amazon ECR プライベートレジストリアクセス許可ポリシーを使用して、IAM エンティティのアクセス許可のスコープをさらに設定できます。詳細については、「レジストリ許可の使用」を参照してください。
- プルスルーキャッシュワークフローを使用して作成された Amazon ECR リポジトリは、他の Amazon ECR リポジトリと同様に処理されます。レプリケーションやイメージスキャンなど、すべてのリポジトリ機能がサポートされています。
- Amazon ECR がプルスルーキャッシュアクションを使用してユーザーに代わって新しいリポジトリを作成すると、一致するリポジトリ作成テンプレートがない限り、次のデフォルト設定がリポジトリに適用されます。リポジトリ作成テンプレートを使用して、Amazon ECR がユーザーに代わって作成したリポジトリに適用される設定を定義できます。詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。

  - タグのイミュータビリティ – タグのイミュータビリティは、イメージタグを上書きできるかどうかを指定します。デフォルトでは、イメージタグはミュータブル (上書き可能) です。タグの動作を変更するには、[ミュータブル] が選択されている場合は [ミュータブルタグ除外] テキストボックスでタグ除外フィルターを設定し、[イミュータブル] が選択されている場合は [イミュータブルタグ除外] テキストボックスでタグ除外フィルターを設定します。
  - 暗号化 – デフォルトの AES256 暗号化が使用されます。
  - リポジトリ権限 – 省略。リポジトリ権限ポリシーは適用されません。
  - ライフサイクルポリシー – 省略。ライフサイクルポリシーは適用されません。
  - リソースタグ – 省略。リソースタグは適用されません。
- プルスルーキャッシュルールを使用してリポジトリのイメージタグのイミュータビリティを有効にすると、Amazon ECR が同じタグを使用してイメージを更新できなくなります。
- プルスルーキャッシュルールを使用して初めてイメージをプルする場合、インターネットへのルートが必要になる場合があります。特定の状況ではインターネットへのルートが必要になるため、障害を回避するためにルートを設定することをお勧めします。したがって、 を使用してインターフェイス VPC エンドポイントを使用するように Amazon ECR を設定している場合は AWS PrivateLink 、最初のプルにインターネットへのルートがあることを確認する必要があります。これを行う方法の 1 つは、同じ VPC にインターネットゲートウェイを持つパブリックサブネットを作成し、プライベートサブネットからインターネットへのすべてのアウトバウンドトラフィックをパブリックサブネットにルーティングすることです。プルスルーキャッシュルールを使用した後続のイメージプルでは、これは必要ありません。詳細については、Amazon Virtual Private Cloud ユーザーガイドの「ルーティングオプションの例」を参照してください。

アップストリームレジストリと Amazon ECR プライベートレジストリを同期するために必要な IAM アクセス許可

プルスルーキャッシュルールを効果的に使用するためには、プライベートレジストリへの認証とイメージのプッシュとプルに必要となる Amazon ECR API のアクセス許可に加えて、次のアクセス許可も更に必要となります。

- `ecr:CreatePullThroughCacheRule` – プルスルーキャッシュルールを作成するアクセス許可を付与します。このアクセス許可は、アイデンティティに基づく IAM ポリシーを介して付与する必要があります。
- `ecr:BatchImportUpstreamImage` – 外部イメージを取得し、プライベートレジストリにインポートするアクセス許可を付与します。このアクセス許可は、プライベートレジストリアクセ

ス許可ポリシー、ID に基づく IAM ポリシー、またはリソースに基づくリポジトリアクセス許可ポリシーを使用して付与できます。レポジトリアクセス許可の使用に関する詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。

- `ecr:CreateRepository` – プライベートレジストリにリポジトリを作成するアクセス許可を付与します。キャッシュされたイメージを格納するリポジトリがまだ存在しない場合には、この許可が必要となります。このアクセス許可は、アイデンティティに基づく IAM ポリシーまたはプライベートレジストリアクセス許可のいずれかによって付与できます。

レジストリ許可の使用

Amazon ECR プライベートレジストリ許可は、プルスルーキャッシュを使用する個々の IAM エンティティのアクセス許可のスコープ設定に使用できます。レジストリ許可ポリシーによって付与されるアクセス許可よりも多くのアクセス許可が IAM ポリシーによって IAM エンティティに付与される場合、IAM ポリシーが優先されます。例えば、 ユーザーに `ecr:*` 許可が付与されている場合には、レジストリレベルで更に許可する必要はありません。

プライベートレジストリの許可ポリシーを作成するには (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ許可ステートメントを設定するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Registry permissions] (レジストリー許可) の順に選択します。
4. [Registry permissions] (レジストリー許可) ページで [Generate statement] (ステートメントを生成) を選択します。
5. 作成するプルスルーキャッシュ許可ポリシーステートメントごとに、次の操作を行います。

    a. [ポリシータイプ] で、[プルスルーキャッシュポリシー] を選択します。

    b. [ステートメント ID] で、プルスルーキャッシュステートメントポリシーの名前を指定します。

    c. [IAM entities] (IAM エンティティ) で、ポリシーに含めるユーザー、グループ、またはロールを指定します。

    d. [リポジトリ名前空間] で、ポリシーを関連付けるプルスルーキャッシュルールを選択します。

e. [リポジトリ名] で、ルールを適用するリポジトリベース名を指定します。たとえば、Amazon ECR パブリックで Amazon Linux リポジトリを指定する場合、リポジトリ名は `amazonlinux` になります。

プライベートレジストリの許可ポリシーを作成するには (AWS CLI)

を使用してプライベートレジストリのアクセス許可を指定するには、次の AWS CLI コマンドを使用します AWS CLI。

1. レジストリポリシーのコンテンツが含まれる `ptc-registry-policy.json` という名のローカルファイルを作成します。次の例では、リポジトリを作成し、以前に作成したプルスルーキャッシュルールに関連付けられたアップストリームソースである Amazon ECR Public からイメージをプルする `ecr-pull-through-cache-user` 許可が付与されています。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "PullThroughCacheFromReadOnlyRole",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/ecr-pull-through-cache-user"
      },
      "Action": [
        "ecr:CreateRepository",
        "ecr:BatchImportUpstreamImage"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/ecr-public/
*"
    }
  ]
}
```

> **Important**
>
> `ecr-CreateRepository` 許可は、キャッシュされたイメージを格納するリポジトリがまだ存在しない場合にのみ必要となります。たとえば、リポジトリ作成アクションとイメージをプルするアクションが、管理者や開発者などの別の IAM プリンシパルによって実行されている場合などです。

2. レジストリポリシーを設定するには、put-registry-policy コマンドを使用します。

```
aws ecr put-registry-policy \
     --policy-text file://ptc-registry.policy.json
```

次の手順

プルスルーキャッシュルールを使用する準備ができたら、次のステップは以下のとおりです。

- プルスルーキャッシュルールを作成します。詳細については、「Amazon ECR でのプルスルーキャッシュルールの作成」を参照してください。
- リポジトリ作成テンプレートを作成します。リポジトリ作成テンプレートは、プルスルーキャッシュアクションで Amazon ECR がユーザーに代わって作成した新しいリポジトリに使用する設定を定義することができます。詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。

クロスアカウント ECR から ECR PTC へのアクセス許可を設定する

Amazon ECR から Amazon ECR (ECR から ECR) へのプルスルーキャッシュ機能を使用すると、リージョン、 AWS アカウント、またはその両方間でイメージを自動的に同期できます。ECR to ECR PTC を使用すると、プライマリ Amazon ECR レジストリにイメージをプッシュし、ダウンストリーム Amazon ECR レジストリにイメージをキャッシュするようにプルスルーキャッシュルールを設定できます。

クロスアカウント ECR から ECR へのプルスルーキャッシュに必要な IAM ポリシー

異なる AWS アカウント間で Amazon ECR レジストリ間でイメージをキャッシュするには、ダウンストリームアカウントに IAM ロールを作成し、このセクションのポリシーを設定して、次のアクセス許可を付与します。

- Amazon ECR には、ユーザーに代わってアップストリーム Amazon ECR レジストリからイメージをプルするためのアクセス許可が必要です。これらのアクセス許可を付与するには、IAM ロールを作成し、プルスルーキャッシュルールで指定します。
- アップストリームレジストリ所有者は、イメージをリソースポリシーにプルするために必要なアクセス許可をキャッシュレジストリ所有者に付与する必要があります。

ポリシー

- プルスルーキャッシュのアクセス許可を定義する IAM ロールを作成する
- IAM ロールの信頼ポリシーを作成する
- アップストリーム Amazon ECR レジストリでリソースポリシーを作成する

プルスルーキャッシュのアクセス許可を定義する IAM ロールを作成する

次の例は、ユーザーに代わってアップストリーム Amazon ECR レジストリからイメージをプルするアクセス許可を IAM ロールに付与するアクセス許可ポリシーを示しています。Amazon ECR がこのロールを引き受ける場合、このポリシーで指定されたアクセス許可を受け取ります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken",
                "ecr:BatchImportUpstreamImage",
                "ecr:BatchGetImage",
```

```
                "ecr:GetImageCopyStatus",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM ロールの信頼ポリシーを作成する

次の例は、ロールを引き受けることができる AWS サービスプリンシパルとして Amazon ECR プルスルーキャッシュを識別する信頼ポリシーを示しています。

JSON

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

アップストリーム Amazon ECR レジストリでリソースポリシーを作成する

アップストリーム Amazon ECR レジストリ所有者は、レジストリポリシーまたはリポジトリポリシーを追加して、ダウンストリームレジストリ所有者に次のアクションを実行するために必要なアクセス許可を付与する必要があります。

> **Note**
>
> 次のリソースポリシーは、クロスアカウント ECR から ECR へのプルスルーキャッシュ設定にのみ必要です。同じアカウント、クロスリージョンのプルスルーキャッシュの場合、前のセクションで示した IAM ロールポリシーと信頼ポリシーのみが必要です。アップストリームレジストリとダウンストリームレジストリが同じアカウントにある場合、ルート AWS アカウントプリンシパルのアクセス許可は必要ありません。

```
{
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::444455556666:root"
    },
    "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchImportUpstreamImage",
        "ecr:GetImageCopyStatus"
    ],
    "Resource": "arn:aws:ecr:region:111122223333:repository/*"
}
```

Amazon ECR でのプルスルーキャッシュルールの作成

Amazon ECR プライベートレジストリにキャッシュするイメージを含むアップストリームレジストリごとに、プルスルーキャッシュルールを作成する必要があります。

シークレットでの認証が必要なアップストリームレジストリでは、認証情報を Secrets Manager シークレットに保存する必要があります。既存の シークレットを使用するか、新しいシークレットを作成できます。Secrets Manager シークレットは、Amazon ECR コンソールで作成することも、Secrets Manager コンソールで作成することもできます。Amazon ECR コンソールの代わりに Secrets Manager コンソールを使用して Secrets Manager シークレットを作成するには、「シー AWS Secrets Manager クレットへのアップストリームリポジトリ認証情報の保存」を参照してください。

前提条件

- プルスルーキャッシュルールを作成するための適切な IAM アクセス許可を持っていることを確認します。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリを同期するために必要な IAM アクセス許可」を参照してください。
- シークレットでの認証が必要なアップストリームレジストリの場合: 既存のシークレットを使用する場合は、Secrets Manager シークレットが以下の要件を満たしていることを確認します。
  - シークレットの名前は `ecr-pullthroughcache/` で始まります。 AWS マネジメントコンソール には、`ecr-pullthroughcache/` プレフィックスが付いた Secrets Manager シークレットのみが表示されます。
  - シークレットが存在するアカウントおよびリージョンは、プルスルーキャッシュルールが存在するアカウントおよびリージョンと一致する必要があります。

プルスルーキャッシュルールの作成 (AWS マネジメントコンソール)

以下のステップは、Amazon ECR コンソールを使用してプルスルーキャッシュルールと Secrets Manager シークレットを作成する方法を示しています。Secrets Manager コンソールを使用してシークレットを作成するには、「シー AWS Secrets Manager クレットへのアップストリームリポジトリ認証情報の保存」を参照してください。

Amazon ECR パブリック、Kubernetes コンテナレジストリ、または Quay の場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. [ステップ 1: ソースを指定] ページの [レジストリ] で、アップストリームレジストリのリストから Amazon ECR パブリック、Kubernetes、または Quay のいずれかを選択し、[次へ] を選択します。
6. [ステップ 2: 宛先を指定する] ページの [Amazon ECR リポジトリプレフィックス] で、ソースパブリックレジストリから取得したイメージをキャッシュするときに使用するリポジトリ名前空間プレフィックスを指定し、[次へ] を選択します。デフォルトでは、名前空間は設定されていますが、カスタム名前空間も指定できます。

7. [ステップ 3: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。
8. 作成する各プルスルーキャッシュに対して、前のステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

Docker Hub の場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. [ステップ 1: ソースを指定] ページの [レジストリ] で [Docker Hub] を選択し、[次へ] を選択します。
6. [ステップ 2: 認証の設定] ページの [アップストリームの認証情報] では、Docker Hub の認証認証情報を AWS Secrets Manager シークレットに保存する必要があります。既存のシークレットを指定するか、Amazon ECR コンソールを使用して新しいシークレットを作成できます。

   a. 既存のシークレットを使用するには、既存のシーク AWS レットを使用するを選択します。[シークレット名] では、ドロップダウンを使用して既存のシークレットを選択し、[次へ] を選択します。

      > **Note**
      >
      > には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。シークレットは、プルスルーキャッシュルールが作成されたのと同じアカウントとリージョンにある必要もあります。

   b. 新しいシークレットを作成するには、[AWS シークレットを作成する] を選択し、次の操作を行って、[次へ] を選択します。

      i. [シークレット名] には、シークレットのわかりやすい名前を指定します。シークレット名は 1～512 文字の Unicode 文字で構成されます。

      ii. [Docker Hub の E メール] には、Docker Hub の E メールを指定します。

iii. [Docker Hub のアクセストークン]には、Docker Hub アクセストークンを指定します。Docker Hub アクセストークンの作成について詳しくは、Docker ドキュメントの「Create and manage access tokens」を参照してください。

7. [ステップ 3: 宛先を指定する] ページの [Amazon ECR リポジトリプレフィックス] で、ソースパブリックレジストリから取得したイメージをキャッシュするときに使用するリポジトリ名前空間を指定し、[次へ] を選択します。

   デフォルトでは、名前空間は設定されていますが、カスタム名前空間も指定できます。

8. [ステップ 4: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。

9. 作成する各プルスルーキャッシュに対して、前のステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

GitHub コンテナレジストリの場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. [ステップ 1: ソースを指定] ページの [レジストリ] で [GitHub コンテナレジストリ] を選択し、[次へ] を選択します。
6. [ステップ 2: 認証の設定] ページの [アップストリームの認証情報] では、GitHub コンテナレジストリ の認証認証情報を AWS Secrets Manager シークレットに保存する必要があります。既存のシークレットを指定するか、Amazon ECR コンソールを使用して新しいシークレットを作成できます。

   a. 既存のシークレットを使用するには、既存のシーク AWS レットを使用するを選択します。[シークレット名] では、ドロップダウンを使用して既存のシークレットを選択し、[次へ] を選択します。

      > **Note**
      >
      > には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。シー

クレットは、プルスルーキャッシュルールが作成されたのと同じアカウントとリージョンにある必要もあります。

b. 新しいシークレットを作成するには、[AWS シークレットを作成する] を選択し、次の操作を行って、[次へ] を選択します。

   i. [シークレット名] には、シークレットのわかりやすい名前を指定します。シークレット名は 1～512 文字の Unicode 文字で構成されます。

   ii. [GitHub コンテナレジストリのユーザー名] には、GitHub コンテナレジストリのユーザー名を指定します。

   iii. [GitHub コンテナレジストリアクストークン] には、GitHub コンテナレジストリアクセストークンを指定します。GitHub アクセストークンの作成の詳細については、GitHub ドキュメントの「個人用アクセストークンを管理する」を参照してください。

7. [ステップ 3: 宛先を指定する] ページの [Amazon ECR リポジトリプレフィックス] で、ソースパブリックレジストリから取得したイメージをキャッシュするときに使用するリポジトリ名前空間を指定し、[次へ] を選択します。

   デフォルトでは、名前空間は設定されていますが、カスタム名前空間も指定できます。

8. [ステップ 4: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。

9. 作成する各プルスルーキャッシュに対して、前のステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

Microsoft Azure コンテナレジストリの場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. [ステップ 1:ソースの指定] ページで、以下の操作を行います。

   a. [レジストリ] には、[Microsoft Azure コンテナレジストリ] を選択します。

   b. [ソースレジストリ URL] に Microsoft Azure コンテナレジストリの名前を指定し、[次へ] を選択します。

> **Important**
>
> `.azurecr.io` サフィックスはユーザーに代わって入力されるため、プレフィックスを指定するだけで済みます。

6. [ステップ 2: 認証の設定] ページの [アップストリームの認証情報] では、Microsoft Azure コンテナレジストリ の認証認証情報を AWS Secrets Manager シークレットに保存する必要があります。既存のシークレットを指定するか、Amazon ECR コンソールを使用して新しいシークレットを作成できます。

   a. 既存のシークレットを使用するには、既存のシーク AWS レットを使用するを選択します。[シークレット名] では、ドロップダウンを使用して既存のシークレットを選択し、[次へ] を選択します。

      > **Note**
      >
      > には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。シークレットは、プルスルーキャッシュルールが作成されたのと同じアカウントとリージョンにある必要もあります。

   b. 新しいシークレットを作成するには、[AWS シークレットを作成する] を選択し、次の操作を行って、[次へ] を選択します。

      i. [シークレット名] には、シークレットのわかりやすい名前を指定します。シークレット名は 1～512 文字の Unicode 文字で構成されます。

      ii. [Microsoft Azure コンテナレジストリのユーザー名] には、Microsoft Azure コンテナーレジストリのユーザー名を指定します。

      iii. [Microsoft Azure コンテナレジストリのアクセストークン] には、Microsoft Azure コンテナレジストリのアクセストークンを指定します。Microsoft Azure コンテナレジストリアクストークンの作成について詳しくは、Microsoft Azure ドキュメントの「トークンを作成する - ポータル」を参照してください。

7. [ステップ 3: 宛先を指定する] ページの [Amazon ECR リポジトリプレフィックス] で、ソースパブリックレジストリから取得したイメージをキャッシュするときに使用するリポジトリ名前空間を指定し、[次へ] を選択します。

   デフォルトでは、名前空間は設定されていますが、カスタム名前空間も指定できます。

8. [ステップ 4: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。
9. 作成する各プルスルーキャッシュに対して、前のステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

GitLab コンテナレジストリの場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. [ステップ 1: ソースを指定] ページの [レジストリ] で、[GitLab コンテナレジストリ] を選択し、[次へ] を選択します。
6. [ステップ 2: 認証の設定] ページの [アップストリーム認証情報] では、GitLab コンテナレジストリの認証情報を AWS Secrets Manager シークレットに保存する必要があります。既存のシークレットを指定するか、Amazon ECR コンソールを使用して新しいシークレットを作成できます。

   a. 既存のシークレットを使用するには、既存の AWS シークレットを使用するを選択します。[シークレット名] では、ドロップダウンを使用して既存のシークレットを選択し、[次へ] を選択します。Secrets Manager コンソールを使用して Secrets Manager シークレットを作成する方法の詳細については、シー AWS Secrets Manager クレットへのアップストリームリポジトリ認証情報の保存 を参照してください。

   > **Note**
   >
   > には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。シークレットは、プルスルーキャッシュルールが作成されたのと同じアカウントとリージョンにある必要もあります。

   b. 新しいシークレットを作成するには、[AWS シークレットを作成する] を選択し、次の操作を行って、[次へ] を選択します。

i. [シークレット名] には、シークレットのわかりやすい名前を指定します。シークレット名は 1～512 文字の Unicode 文字で構成されます。

ii. [GitLab コンテナレジストリのユーザー名] には、GitLab コンテナレジストリのユーザー名を指定します。

iii. [GitLab コンテナレジストリのアクセストークン] には、GitLab コンテナレジストリのアクセストークンを指定します。GitLab コンテナレジストリのアクセストークンの作成の詳細については、GitLab ドキュメントの「Personal access tokens」、「Group access tokens」、または「Project access tokens」を参照してください。

7. [ステップ 3: 宛先を指定する] ページの [Amazon ECR リポジトリプレフィックス] で、ソースパブリックレジストリから取得したイメージをキャッシュするときに使用するリポジトリ名前空間を指定し、[次へ] を選択します。

   デフォルトでは、名前空間は設定されていますが、カスタム名前空間も指定できます。

8. [ステップ 4: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。

9. 作成する各プルスルーキャッシュに対して、前のステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

AWS アカウント内の Amazon ECR プライベートレジストリの場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. [ステップ 1: アップストリームの指定] ページの [レジストリ] で、[Amazon ECR Private] と [このアカウント] を選択します。[リージョン] で、アップストリーム Amazon ECR レジストリのリージョンを選択し、[次へ] を選択します。
6. [ステップ 2: 名前空間の指定] ページで、[キャッシュ名前空間] にプルスルーキャッシュレポジトリの作成方法 ([特定のプレフィックス] またはプレフィックスなし) を選択します。[特定のプレフィックス] を選択した場合、アップストリームレジストリからイメージをキャッシュするための名前空間の一部として使用するプレフィックス名を指定する必要があります。

7. [アップストリーム名前空間] で、アップストリームレジストリに存在する [特定のプレフィックス] からプルするかどうかを選択します。プレフィックスなしを選択した場合、アップストリームレジストリ内の任意のリポジトリからプルできます。プロンプトが表示されたら、アップストリームリポジトリプレフィックスを指定し、[次へ] を選択します。

   > **Note**
   >
   > キャッシュおよびアップストリーム名前空間のカスタマイズの詳細については、「ECR から ECR へのプルスルーキャッシュのリポジトリプレフィックスをカスタマイズする」を参照してください。

8. [ステップ 3: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。
9. 作成する各プルスルーキャッシュに対して、このステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

別の AWS アカウントの Amazon ECR プライベートレジストリの場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. [ステップ 1: アップストリームの指定] ページの [レジストリ] で、[Amazon ECR Private] と [クロスアカウント] を選択します。[リージョン] で、アップストリーム Amazon ECR レジストリのリージョンを選択します。Account で、アップストリーム Amazon ECR レジストリの AWS アカウント ID を指定し、Next を選択します。
6. [ステップ 2: アクセス許可の指定] ページで、[IAM ロール]にクロスアカウントプルスルーキャッシュアクセスに使用するロールを選択し、[作成] を選択します。

   > **Note**
   >
   > クロスアカウント ECR から ECR へのプルスルーキャッシュに必要な IAM ポリシー で作成されたアクセス許可を使用する IAM ロールを選択してください。

7. [ステップ 3: 名前空間の指定] ページで、[キャッシュ名前空間] にプルスルーキャッシュレポジトリの作成方法 ([特定のプレフィックス] またはプレフィックスなし) を選択します。[特定のプレフィックス] を選択した場合、アップストリームレジストリからイメージをキャッシュするための名前空間の一部として使用するプレフィックス名を指定する必要があります。
8. [アップストリーム名前空間] で、アップストリームレジストリに存在する [特定のプレフィックス] からプルするかどうかを選択します。プレフィックスなしを選択した場合、アップストリームレジストリ内の任意のリポジトリからプルできます。プロンプトが表示されたら、アップストリームリポジトリプレフィックスを指定し、[次へ] を選択します。

> **Note**
>
> キャッシュおよびアップストリーム名前空間のカスタマイズの詳細については、「ECR から ECR へのプルスルーキャッシュのリポジトリプレフィックスをカスタマイズする」を参照してください。

9. [ステップ 4: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。
10. 作成する各プルスルーキャッシュに対して、このステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

Chainguard レジストリの場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プライベートレジストリ設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [Pull through cache configuration] (プルスルーキャッシュの設定) ページで、[Add rule] (ルールの追加) を選択します。
5. ステップ 1: ソースページを指定する、レジストリで、Chainguard Registry, Next を選択します。
6. ステップ 2: 認証を設定するページで、アップストリーム認証情報用に、Chainguard Registry の認証情報を AWS Secrets Manager シークレットに保存する必要があります。既存のシークレットを指定するか、Amazon ECR コンソールを使用して新しいシークレットを作成できます。

   a. 既存のシークレットを使用するには、既存のシーク AWS レットを使用するを選択します。[シークレット名] では、ドロップダウンを使用して既存のシークレットを選択し、[次

へ] を選択します。Secrets Manager コンソールを使用して Secrets Manager シークレットを作成する方法の詳細については、シー AWS Secrets Manager クレットへのアップストリームリポジトリ認証情報の保存 を参照してください。

> **Note**
>
> には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。シークレットは、プルスルーキャッシュルールが作成されたのと同じアカウントとリージョンにある必要もあります。

b. 新しいシークレットを作成するには、[AWS シークレットを作成する] を選択し、次の操作を行って、[次へ] を選択します。

   i. [シークレット名] には、シークレットのわかりやすい名前を指定します。シークレット名は 1～512 文字の Unicode 文字で構成されます。

   ii. Chainguard Registry ユーザー名には、Chainguard Registry ユーザー名を指定します。

   iii. Chainguard Registry プルトークンには、Chainguard Registry プルトークンを指定します。Chainguard Registry プルトークンの作成の詳細については、Chainguard ドキュメントの「プルトークンによる認証」を参照してください。

7. ステップ 3: 送信先ページの Amazon ECR リポジトリプレフィックスで、ソースレジストリからプルされたイメージをキャッシュするときに使用するリポジトリ名前空間を指定し、次へを選択します。

   デフォルトでは、名前空間は設定されていますが、カスタム名前空間も指定できます。

8. [ステップ 4: 確認と作成] ページで、プルスルーキャッシュルールの設定を確認し、[作成] を選択します。

9. 作成する各プルスルーキャッシュに対して、前のステップを繰り返します。プルスルーキャッシュルールは、リージョンごとに個別に作成されます。

プルスルーキャッシュルールを作成するには (AWS CLI)

create-pull-through-cache-rule AWS CLI コマンドを使用して、Amazon ECR プライベートレジストリのプルスルーキャッシュルールを作成します。シークレットでの認証が必要なアップストリームレジストリでは、認証情報を Secrets Manager シークレットに保存する必要があります。Secrets

Manager コンソールを使用してシークレットを作成するには、「シー AWS Secrets Manager クレットへのアップストリームリポジトリ認証情報の保存」を参照してください。

以下の例は、サポートされている各アップストリームレジストリについて提供されています。

Amazon ECR Public の場合

次の例では、Amazon ECR パブリックレジストリのプルスルーキャッシュルールを作成します。リポジトリプレフィックス ecr-public を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム ecr-public/*upstream-repository-name* を持ちます。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix ecr-public \
     --upstream-registry-url public.ecr.aws \
     --region us-east-2
```

Kubernetes コンテナレジストリの場合

次の例では、Kubernetes パブリックレジストリのプルスルーキャッシュルールを作成します。リポジトリプレフィックス kubernetes を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム kubernetes/*upstream-repository-name* を持ちます。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix kubernetes \
     --upstream-registry-url registry.k8s.io \
     --region us-east-2
```

Quay の場合

次の例では、Quay パブリックレジストリのプルスルーキャッシュルールを作成します。リポジトリプレフィックス quay を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム quay/*upstream-repository-name* を持ちます。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix quay \
     --upstream-registry-url quay.io \
     --region us-east-2
```

Docker Hub の場合

次の例では、Docker Hub レジストリのプルスルーキャッシュルールを作成します。リポジトリプレフィックス docker-hub を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム docker-hub/*upstream-repository-name* を持ちます。Docker Hub の認証情報が含まれているシークレットの完全な Amazon リソースネーム (ARN) を指定する必要があります。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix docker-hub \
     --upstream-registry-url registry-1.docker.io \
     --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-2
```

GitHub コンテナレジストリの場合

次の例では、GitHub コンテナレジストリ用のプルスルーキャッシュルールを作成します。リポジトリプレフィックス github を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム github/*upstream-repository-name* を持ちます。GitHub コンテナレジストリの認証情報が含まれているシークレットの完全な Amazon リソースネーム (ARN) を指定する必要があります。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix github \
     --upstream-registry-url ghcr.io \
     --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-2
```

Microsoft Azure コンテナレジストリの場合

次の例では、Microsoft Azure コンテナレジストリ用のプルスルーキャッシュルールを作成します。リポジトリプレフィックス azure を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム azure/*upstream-repository-name* を持ちます。Microsoft Azure コンテナレジストリの認証情報が含まれているシークレットの完全な Amazon リソースネーム (ARN) を指定する必要があります。

```
aws ecr create-pull-through-cache-rule \
```

```
    --ecr-repository-prefix azure \
    --upstream-registry-url myregistry.azurecr.io \
    --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
    --region us-east-2
```

GitLab コンテナレジストリの場合

次の例では、GitLab コンテナレジストリ用のプルスルーキャッシュルールを作成します。リポジトリプレフィックス gitlab を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム gitlab/*upstream-repository-name* を持ちます。GitLab コンテナレジストリの認証情報が含まれているシークレットの完全な Amazon リソースネーム (ARN) を指定する必要があります。

```
aws ecr create-pull-through-cache-rule \
    --ecr-repository-prefix gitlab \
    --upstream-registry-url registry.gitlab.com \
    --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
    --region us-east-2
```

AWS アカウント内の Amazon ECR プライベートレジストリの場合

次の の例では、同じ AWS アカウント内のクロスリージョンの Amazon ECR プライベートレジストリのプルスルーキャッシュルールを作成します。リポジトリプレフィックス ecr を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム ecr/*upstream-repository-name* を持ちます。

```
aws ecr create-pull-through-cache-rule \
    --ecr-repository-prefix ecr \
    --upstream-registry-url aws_account_id.dkr.ecr.region.amazonaws.com \
    --region us-east-2
```

別の AWS アカウントの Amazon ECR プライベートレジストリの場合

次の の例では、同じ AWS アカウント内のクロスリージョンの Amazon ECR プライベートレジストリのプルスルーキャッシュルールを作成します。リポジトリプレフィックス ecr を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム ecr/*upstream-repository-name* を持ちます。IAM ロールの完全な Amazon リソースネーム

(ARN) と、Amazon ECR でのプルスルーキャッシュルールの作成 で作成されたアクセス許可を指定する必要があります。

```
aws ecr create-pull-through-cache-rule \
 --ecr-repository-prefix ecr \
 --upstream-registry-url aws_account_id.dkr.ecr.region.amazonaws.com \
 --custom-role-arn arn:aws:iam::aws_account_id:role/example-role \
 --region us-east-2
```

Chainguard レジストリの場合

次の例では、Chainguard Registry のプルスルーキャッシュルールを作成します。リポジトリプレフィックス chainguard を指定します。この結果、プルスルーキャッシュルールを使用して作成された各リポジトリは命名スキーム chainguard/*upstream-repository-name* を持ちます。Chainguard Registry 認証情報を含むシークレットの完全な Amazon リソースネーム (ARN) を指定する必要があります。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix chainguard \
     --upstream-registry-url cgr.dev \
     --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-2
```

次の手順

プルスルーキャッシュルールを作成したら、次のステップは以下のとおりです。

- リポジトリ作成テンプレートを作成します。リポジトリ作成テンプレートは、プルスルーキャッシュアクションで Amazon ECR がユーザーに代わって作成した新しいリポジトリに使用する設定を定義することができます。詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。
- プルスルーキャッシュルールを検証します。プルスルーキャッシュルールを検証する際、Amazon ECR はアップストリームレジストリとのネットワーク接続を確立し、アップストリームレジストリの認証情報を含む Secrets Manager シークレットにアクセスできること、および認証が成功したことを確認します。詳細については、「Amazon ECR でのプルスルーキャッシュルールの検証」を参照してください。

- プルスルーキャッシュルールの使用を開始します。詳細については、「Amazon ECR でのプルスルーキャッシュルールを使用したイメージのプル」を参照してください。

Amazon ECR でのプルスルーキャッシュルールの検証

プルスルーキャッシュルールを作成した後、認証が必要なアップストリームレジストリに対して、そのルールが適切に機能しているかどうかを検証できます。プルスルーキャッシュルールを検証する際、Amazon ECR はアップストリームレジストリとのネットワーク接続を確立し、アップストリームレジストリの認証情報を含む Secrets Manager シークレットにアクセスできること、および認証が成功したことを確認します。

プルスルーキャッシュルールの使用を開始する前に、適切な IAM アクセス許可を持っていることを確認してください。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリを同期するために必要な IAM アクセス許可」を参照してください。

プルスルーキャッシュルールを検証するには (AWS マネジメントコンソール)

以下のステップは、Amazon ECR コンソールを使用してプルスルーキャッシュルールを検証する方法を示しています。

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、検証するプルスルーキャッシュルールが含まれているリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ)、[Pull through cache] (プルスルーキャッシュ) の順に選択します。
4. [プルスルーキャッシュ設定] ページで、検証するプルスルーキャッシュルールを選択します。次に、[アクション] ドロップダウンメニューを使用して [詳細を表示] を選択します。
5. プルスルーキャッシュルールの詳細ページで、[アクション] ドロップダウンメニューを使用して [認証を確認] を選択します。Amazon ECR は結果を示すバナーを表示します。
6. 検証する各プルスルーキャッシュルールに対して、これらのステップを繰り返します。

プルスルーキャッシュルールを検証するには (AWS CLI)

validate-pull-through-cache-rule AWS CLI コマンドは、Amazon ECR プライベートレジストリのプルスルーキャッシュルールを検証するために使用されます。次の例では `ecr-public` 名前空間プレフィックスを使用しています。その値を、検証するプルスルーキャッシュルールのプレフィックス値に置き換えます。

```
aws ecr validate-pull-through-cache-rule \
     --ecr-repository-prefix ecr-public \
     --region us-east-2
```

レスポンスで、isValid パラメータは検証が成功したかどうかを示します。true の場合、Amazon ECR はアップストリームレジストリにアクセスでき、認証は成功しました。false の場合、問題が発生し、検証が失敗しました。failure パラメータは原因を示します。

Amazon ECR でのプルスルーキャッシュルールを使用したイメージのプル

以下の例は、プルスルーキャッシュルールを使用してイメージをプルするときに使用するコマンド構文を示しています。プルスルーキャッシュルールを使用してアップストリームイメージをプルする際にエラーが発生した場合は、Amazon ECR でのプルスルーキャッシュ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。最も一般的なエラーと、それらを解決する方法が記載されています。

プルスルーキャッシュルールの使用を開始する前に、適切な IAM アクセス許可を持っていることを確認してください。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリを同期するために必要な IAM アクセス許可」を参照してください。

> **Note**
>
> 次の例では、 が AWS マネジメントコンソール 使用するデフォルトの Amazon ECR リポジトリ名前空間値を使用します。設定した Amazon ECR プライベートリポジトリ URI を使用していることを確認します。

Amazon ECR Public の場合

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/ecr-public/repository_name/
image_name:tag
```

Kubernetes コンテナレジストリ

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/kubernetes/repository_name/
image_name:tag
```

Quay

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/quay/repository_name/
image_name:tag
```

Docker Hub

Docker Hub の公式イメージの場合:

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/docker-hub/
library/image_name:tag
```

> **Note**
>
> Docker Hub 公式イメージの場合は、/library プレフィックスを含める必要があります。その他すべての Docker Hub リポジトリでは、/library プレフィックスを省略する必要があります。

その他すべての Docker Hub イメージの場合:

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/docker-hub/repository_name/
image_name:tag
```

GitHub コンテナレジストリ

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/github/repository_name/
image_name:tag
```

Microsoft Azure コンテナレジストリ

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/azure/repository_name/
image_name:tag
```

GitLab コンテナレジストリ

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/gitlab/repository_name/
image_name:tag
```

Chainguard レジストリ

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/chainguard/repository_name/
image_name:tag
```

シー AWS Secrets Manager クレットへのアップストリームリポジトリ認証情報の保存

認証を必要とするアップストリームリポジトリのプルスルーキャッシュルールを作成する場合、認証情報を Secrets Manager シークレットに保存する必要があります。Secrets Manager シークレットを使用するとコストがかかる場合があります。詳細については、AWS Secrets Manager 料金表を参照してください。

以下の手順では、サポートされている各アップストリームリポジトリの Secrets Manager シークレットを作成する方法を示します。Secrets Manager コンソールを使用してシークレットを作成する代わりに、オプションで Amazon ECR コンソールのプルスルーキャッシュルール作成ワークフローを使用してシークレットを作成できます。詳細については、「Amazon ECR でのプルスルーキャッシュルールの作成」を参照してください。

Docker Hub

Docker Hub 認証情報の Secrets Manager シークレットを作成するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [シークレットのタイプを選択] ページで、次の操作を行います。
    a. [Secret type] (シークレットタイプ) で、[Other type of secret] (他の種類のシークレット) を選択します。
    b. [キー/値のペア] では、Docker Hub 認証情報用に 2 行を作成します。シークレットには最大 65536 バイトまで保存できます。
        i. 最初のキーと値のペアでは、`username` をキーとして指定し、値として Docker Hub のユーザー名を指定します。

ii. 2 番目のキーと値のペアでは、accessToken をキーとして指定し、値として Docker Hub のアクセストークンを指定します。Docker Hub アクセストークンの作成について詳しくは、Docker ドキュメントの「Create and manage access tokens」を参照してください。

c. [暗号化キー] にはデフォルト aws/secretsmanager AWS KMS key 値を選択した状態で、[次へ] を選択します。このキーを使用してもコストは発生しません。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager でのシークレットの暗号化と復号化」を参照してください。

> **Important**
>
> シークレットを暗号化するには、デフォルトの aws/secretsmanager 暗号化キーを使用する必要があります。Amazon ECR では、このためのカスタマーマネージドキー (CMK) の使用はサポートされていません。

4. [シークレットを設定] ページで、次の操作を行います。

a. わかりやすいシークレット名と説明を入力します。シークレット名は 1～512 文字の Unicode 文字を含み、ecr-pullthroughcache/ をプレフィックスとする必要があります。

> **Important**
>
> Amazon ECR には、 ecr-pullthroughcache/ プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。

b. (オプション) [Tags] (タグ) セクションで、タグをシークレットに追加します。タグ付け方法については、AWS Secrets Manager ユーザーガイドの「Tag Secrets Manager シークレット」を参照してください。機密情報は暗号化されていないため、タグに保存しないでください。

c. (オプション) [Resource permissions] (リソースに対するアクセス許可) でリソースポリシーをシークレットに追加するには、[Edit permissions] (アクセス許可の編集) をクリックします。詳細については、AWS Secrets Manager ユーザーガイドの「アクセス許可ポリシーを Secrets Manager シークレットにアタッチする」を参照してください。

d. (オプション) シークレットのレプリケートで、シークレットを別のシークレットにレプリケートするには AWS リージョン、シークレットのレプリケートを選択します。シー

クレットのレプリケーションは、この段階で実行することも、後に戻ってきて実行することもできます。詳細については、AWS Secrets Manager ユーザーガイドの「シークレットを他のリージョンにレプリケートする」を参照してください。

e. [次へ] を選択します。

5. (オプション) [Configure rotation] (ローテーションを設定する) ページで、シークレットの自動ローテーションを有効にできます。ローテーションをオフにしておいて、後でオンにすることもできます。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager シークレットのローテーション」を参照してください。[次へ] を選択します。

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

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

GitHub Container Registry

GitHub コンテナレジストリの認証情報用の Secrets Manager シークレットを作成するには (AWS マネジメントコンソール)

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

   a. [Secret type] (シークレットタイプ) で、[Other type of secret] (他の種類のシークレット) を選択します。

   b. [キー/値のペア] では、GitHub 認証情報用に 2 行を作成します。シークレットには最大 65536 バイトまで保存できます。

      i. 最初のキーと値のペアでは、`username` をキーとして指定し、値として GitHub のユーザー名を指定します。

      ii. 2 番目のキーと値のペアでは、`accessToken` をキーとして指定し、値として GitHub のアクセストークンを指定します。GitHub アクセストークンの作成の詳細については、GitHub ドキュメントの「個人用アクセストークンを管理する」を参照してください。

   c. [暗号化キー] にはデフォルト aws/secretsmanager AWS KMS key 値を選択した状態で、[次へ] を選択します。このキーを使用してもコストは発生しません。詳細について

は、AWS Secrets Manager ユーザーガイドの「Secrets Manager でのシークレットの暗号化と復号化」を参照してください。

> **Important**
>
> シークレットを暗号化するには、デフォルトの `aws/secretsmanager` 暗号化キーを使用する必要があります。Amazon ECR では、このためのカスタマーマネージドキー (CMK) の使用はサポートされていません。

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

   a. わかりやすいシークレット名と説明を入力します。シークレット名は 1～512 文字の Unicode 文字を含み、`ecr-pullthroughcache/` をプレフィックスとする必要があります。

   > **Important**
   >
   > Amazon ECR には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。

   b. (オプション) [Tags] (タグ) セクションで、タグをシークレットに追加します。タグ付け方法については、AWS Secrets Manager ユーザーガイドの「Tag Secrets Manager シークレット」を参照してください。機密情報は暗号化されていないため、タグに保存しないでください。

   c. (オプション) [Resource permissions] (リソースに対するアクセス許可) でリソースポリシーをシークレットに追加するには、[Edit permissions] (アクセス許可の編集) をクリックします。詳細については、AWS Secrets Manager ユーザーガイドの「アクセス許可ポリシーを Secrets Manager シークレットにアタッチする」を参照してください。

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

   e. [次へ] を選択します。

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

ることもできます。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager シークレットのローテーション」を参照してください。[次へ] を選択します。

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

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

Microsoft Azure Container Registry

Microsoft Azure コンテナレジストリの認証情報用の Secrets Manager シークレットを作成するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [シークレットのタイプを選択] ページで、次の操作を行います。
   a. [Secret type] (シークレットタイプ) で、[Other type of secret] (他の種類のシークレット) を選択します。
   b. [キー/値のペア] では、Microsoft Azure 認証情報用に 2 行を作成します。シークレットには最大 65536 バイトまで保存できます。
      i. 最初のキーと値のペアでは、`username` をキーとして指定し、値として Microsoft Azure コンテナレジストリのユーザー名を指定します。
      ii. 2 番目のキーと値のペアでは、`accessToken` をキーとして指定し、値として Microsoft Azure コンテナレジストリのアクセストークンを指定します。Microsoft Azure アクセストークンの作成について詳しくは、Microsoft Azure ドキュメントの「トークンを作成する - ポータル」を参照してください。
   c. [暗号化キー] にはデフォルト aws/secretsmanager AWS KMS key 値を選択した状態で、[次へ] を選択します。このキーを使用してもコストは発生しません。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager でのシークレットの暗号化と復号化」を参照してください。

> **Important**
>
> シークレットを暗号化するには、デフォルトの `aws/secretsmanager` 暗号化キーを使用する必要があります。Amazon ECR では、このためのカスタマーマネージドキー (CMK) の使用はサポートされていません。

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

   a. わかりやすいシークレット名と説明を入力します。シークレット名は 1～512 文字の Unicode 文字を含み、`ecr-pullthroughcache/` をプレフィックスとする必要があります。

   > **Important**
   >
   > Amazon ECR には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。

   b. (オプション) [Tags] (タグ) セクションで、タグをシークレットに追加します。タグ付け方法については、AWS Secrets Manager ユーザーガイドの「Tag Secrets Manager シークレット」を参照してください。機密情報は暗号化されていないため、タグに保存しないでください。

   c. (オプション) [Resource permissions] (リソースに対するアクセス許可) でリソースポリシーをシークレットに追加するには、[Edit permissions] (アクセス許可の編集) をクリックします。詳細については、AWS Secrets Manager ユーザーガイドの「アクセス許可ポリシーを Secrets Manager シークレットにアタッチする」を参照してください。

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

   e. [次へ] を選択します。

5. (オプション) [Configure rotation] (ローテーションを設定する) ページで、シークレットの自動ローテーションを有効にできます。ローテーションをオフにしておいて、後でオンにすることもできます。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager シークレットのローテーション」を参照してください。[次へ] を選択します。

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

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

GitLab Container Registry

GitLab コンテナレジストリの認証情報用の Secrets Manager シークレットを作成する手順 (AWS マネジメントコンソール)

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

   a. [Secret type] (シークレットタイプ) で、[Other type of secret] (他の種類のシークレット) を選択します。

   b. [キー/値のペア] で、GitLab 認証情報用の 2 つの行を作成します。シークレットには最大 65536 バイトまで保存できます。

      i. 最初のキー/値のペアでは、キーとして `username` を指定し、値として GitLab コンテナレジストリのユーザー名を指定します。

      ii. 2 番目のキー/値のペアでは、キーとして `accessToken` を指定し、値として GitLab コンテナレジストリのアクセストークンを指定します。GitLab コンテナレジストリのアクセストークンの作成の詳細については、GitLab ドキュメントの「Personal access tokens」、「Group access tokens」、または「Project access tokens」を参照してください。

   c. [暗号化キー] にはデフォルト aws/secretsmanager AWS KMS key 値を選択した状態で、[次へ] を選択します。このキーを使用してもコストは発生しません。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager でのシークレットの暗号化と復号化」を参照してください。

> **Important**
>
> シークレットを暗号化するには、デフォルトの `aws/secretsmanager` 暗号化キーを使用する必要があります。Amazon ECR では、このためのカスタマーマネージドキー (CMK) の使用はサポートされていません。

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

   a. わかりやすいシークレット名と説明を入力します。シークレット名は 1～512 文字の Unicode 文字を含み、`ecr-pullthroughcache/` をプレフィックスとする必要があります。

   > **Important**
   >
   > Amazon ECR には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。

   b. (オプション) [Tags] (タグ) セクションで、タグをシークレットに追加します。タグ付け方法については、AWS Secrets Manager ユーザーガイドの「Tag Secrets Manager シークレット」を参照してください。機密情報は暗号化されていないため、タグに保存しないでください。

   c. (オプション) [Resource permissions] (リソースに対するアクセス許可) でリソースポリシーをシークレットに追加するには、[Edit permissions] (アクセス許可の編集) をクリックします。詳細については、AWS Secrets Manager ユーザーガイドの「アクセス許可ポリシーを Secrets Manager シークレットにアタッチする」を参照してください。

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

   e. [次へ] を選択します。

5. (オプション) [Configure rotation] (ローテーションを設定する) ページで、シークレットの自動ローテーションを有効にできます。ローテーションをオフにしておいて、後でオンにすることもできます。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager シークレットのローテーション」を参照してください。[次へ] を選択します。

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

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

Chainguard Registry

Chainguard 認証情報の Secrets Manager シークレットを作成するには (AWS マネジメントコンソール)

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

   a. [Secret type] (シークレットタイプ) で、[Other type of secret] (他の種類のシークレット) を選択します。

   b. キーと値のペアで、Chainguard 認証情報の 2 行を作成します。シークレットには最大 65536 バイトまで保存できます。

      i. 最初のキーと値のペアでは、キー`username`として を指定し、値として Chainguard Registry ユーザー名を指定します。

      ii. 2 番目のキーと値のペアでは、キー`accessToken`として を指定し、値として Chainguard Registry アクセストークンを指定します。Chainguard Registry プルトークンの作成の詳細については、Chainguard ドキュメントの「プルトークンによる認証」を参照してください。

   c. [暗号化キー] にはデフォルト aws/secretsmanager AWS KMS key 値を選択した状態で、[次へ] を選択します。このキーを使用してもコストは発生しません。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager でのシークレットの暗号化と復号化」を参照してください。

   > **Important**
   >
   > シークレットを暗号化するには、デフォルトの `aws/secretsmanager` 暗号化キーを使用する必要があります。Amazon ECR では、このためのカスタマーマネージドキー (CMK) の使用はサポートされていません。

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

   a. わかりやすいシークレット名と説明を入力します。シークレット名は 1～512 文字の Unicode 文字を含み、`ecr-pullthroughcache/` をプレフィックスとする必要があります。

   > **Important**
   >
   > Amazon ECR には、 `ecr-pullthroughcache/` プレフィックスを使用する名前の Secrets Manager シークレット AWS マネジメントコンソール のみが表示されます。

   b. (オプション) [Tags] (タグ) セクションで、タグをシークレットに追加します。タグ付け方法については、AWS Secrets Manager ユーザーガイドの「Tag Secrets Manager シークレット」を参照してください。機密情報は暗号化されていないため、タグに保存しないでください。

   c. (オプション) [Resource permissions] (リソースに対するアクセス許可) でリソースポリシーをシークレットに追加するには、[Edit permissions] (アクセス許可の編集) をクリックします。詳細については、AWS Secrets Manager ユーザーガイドの「アクセス許可ポリシーを Secrets Manager シークレットにアタッチする」を参照してください。

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

   e. [次へ] を選択します。

5. (オプション) [Configure rotation] (ローテーションを設定する) ページで、シークレットの自動ローテーションを有効にできます。ローテーションをオフにしておいて、後でオンにすることもできます。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager シークレットのローテーション」を参照してください。[次へ] を選択します。

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

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

ECR から ECR へのプルスルーキャッシュのリポジトリプレフィックスをカスタマイズする

プルスルーキャッシュルールは、ecr リポジトリプレフィックスとアップストリームリポジトリプレフィックスの両方をサポートします。ecr リポジトリプレフィックスは、ルールに関連付けられている Amazon ECR キャッシュレジストリのリポジトリ名前空間プレフィックスです。このプレフィックスを使用するすべてのリポジトリは、ルールで定義されたアップストリームレジストリのキャッシュ対応リポジトリをプルスルーします。例えば、prod というプレフィックスは、prod/ で始まるすべてのリポジトリに適用されます。プルスルーキャッシュルールが関連付けられていないレジストリ内のすべてのリポジトリにテンプレートを適用するには、プレフィックスとして ROOT を使用できます。

> **Important**
>
> プレフィックスの末尾には常に / が適用されると想定されます。ecr-public をプレフィックスとして指定すると、Amazon ECR はそれを ecr-public/ として扱います。

アップストリームリポジトリプレフィックスは、アップストリームリポジトリ名と一致します。デフォルトでは、ROOT に設定され、アップストリームリポジトリとの照合が可能になります。アップストリームリポジトリプレフィックスは、Amazon ECR リポジトリプレフィックスに ROOT 以外の値がある場合にのみ設定できます。

次の表は、プルスルーキャッシュルールのプレフィックス設定に基づくキャッシュリポジトリ名とアップストリームリポジトリ名のマッピングを示しています。

キャッシュ名前空間	アップストリーム名前空間	マッピング関係 (キャッシュリポジトリ → アップストリームリポジトリ)
ecr-public	ROOT (デフォルト)	ecr-public/my-app/image1 → my-app/image1 ecr-public/my-app/image2 → my-app/image2

キャッシュ名前空間	アップストリーム名前空間	マッピング関係 (キャッシュリポジトリ → アップストリームリポジトリ)
ルート	ルート	`my-app/image1` → `my-app/image1`
team-a	team-a	`team-a/myapp/image1` → `team-a/myapp/image1`
my-app	upstream-app	`my-app/image1` → `upstream-app/image1`

Amazon ECR でのプルスルーキャッシュに関する問題のトラブルシューティング

プルスルーキャッシュルールを使用してアップストリームイメージをプルする際に、最もよく発生する可能性のある一般的なエラーは次のとおりです。

リポジトリが存在しない

リポジトリが存在しないことを示すエラーは、ほとんどの場合、Amazon ECR プライベートレジストリにリポジトリが存在しないか、アップストリームイメージをプルする IAM プリンシパルに `ecr:CreateRepository` アクセス許可が付与されていないのが原因です。このエラーを解決するには、プルコマンドのリポジトリ URI が正しいこと、必要となる IAM アクセス許可がアップストリームイメージをプルする IAM プリンシパルに付与されていること、またはアップストリームのイメージをプルする前に、プッシュされるアップストリームイメージのリポジトリが Amazon ECR プライベートレジストリに作成されていることを確認します。必要となる IAM アクセス許可の詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリを同期するために必要な IAM アクセス許可」を参照してください。

このエラーの例を以下に示します。

```
Error response from daemon: repository 111122223333.dkr.ecr.us-east-1.amazonaws.com/
ecr-public/amazonlinux/amazonlinux not found: name unknown: The repository with
 name 'ecr-public/amazonlinux/amazonlinux' does not exist in the registry with id
 '111122223333'
```

リクエストされたイメージが見つからない

イメージが見つからないことを示すエラーは、ほとんどの場合、アップストリームレジストリにイメージが存在しないか、アップストリームイメージをプルする IAM プリンシパルに `ecr:BatchImportUpstreamImage` アクセス許可が付与されておらず、Amazon ECR プライベートレジストリ内にレポジトリがすでに作成されているのが原因です。このエラーを解決するには、アップストリームイメージとイメージタグ名が正しく、それが存在し、アップストリームイメージをプルする IAM プリンシパルに必要な IAM アクセス許可が付与されていることを確認する必要があります。必要となる IAM アクセス許可の詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリを同期するために必要な IAM アクセス許可」を参照してください。

このエラーの例を以下に示します。

```
Error response from daemon: manifest for 111122223333.dkr.ecr.us-
east-1.amazonaws.com/ecr-public/amazonlinux/amazonlinux:latest not found: manifest
 unknown: Requested image not found
```

Docker Hub リポジトリからプルするときに 403 Forbidden

Docker Official Image としてタグ付けされている Docker Hub リポジトリからプルする場合は、使用する URI に `/library/` を含める必要があります。例えば、*aws_account_id*`.dkr.ecr.`*region*`.amazonaws.com/docker-hub/`**`library/`***image_name:tag*。Docker Hub Official イメージの `/library/` を省略した場合、プルスルーキャッシュルールを使用してイメージをプルしようとすると `403 Forbidden` エラーが返されます。詳細については、「Amazon ECR でのプルスルーキャッシュルールを使用したイメージのプル」を参照してください。

このエラーの例を以下に示します。

```
Error response from daemon: failed to resolve reference "111122223333.dkr.ecr.us-
west-2.amazonaws.com/docker-hub/amazonlinux:2023": pulling from host
 111122223333.dkr.ecr.us-west-2.amazonaws.com failed with status code [manifests
 2023]: 403 Forbidden
```

Amazon ECR でのプライベートイメージレプリケーション

リポジトリのレプリケーションをサポートするために、Amazon ECR プライベートレジストリを構成できます。Amazon ECR は、クロスリージョンレプリケーションおよびクロスアカウントレプリケーションの両方をサポートしています。クロスアカウントレプリケーションを実行するには、ターゲットアカウントで、ソースレジストリからのレプリケーションを許可するレジストリのアクセス許可ポリシーを設定する必要があります。詳細については、「Amazon ECR でのプライベートレジストリのアクセス許可」を参照してください。

トピック

- クロスアカウントレプリケーションポリシーの要件
- プライベートイメージのレプリケーションに関する考慮事項
- Amazon ECR でのプライベートイメージレプリケーションの例
- Amazon ECR でのプライベートイメージレプリケーションの設定
- Amazon ECR でのプライベートイメージレプリケーション設定の削除

クロスアカウントレプリケーションポリシーの要件

クロスアカウント ECR レプリケーションを正しく機能させるには、どのアカウントでどのポリシーが設定されているかを理解する必要があります。このセクションでは、送信元アカウントと送信先アカウントの両方のポリシー要件について説明します。

ポリシー設定の概要

クロスアカウント ECR レプリケーションでは、送信先アカウントのみのポリシー設定が必要です。ソースアカウントには、特別なリポジトリポリシーやレジストリポリシーは必要ありません。

- ソースアカウント: レジストリ設定でレプリケーションルールを設定します。ソースリポジトリに追加のポリシーは必要ありません。
- 送信先アカウント: ソースアカウントにイメージのレプリケートを許可するレジストリアクセス許可ポリシーを設定します。

送信先レジストリポリシーの要件

送信先アカウントは、次のアクションを実行するアクセス許可をソースアカウントに付与するレジストリアクセス許可ポリシーを設定する必要があります。

- `ecr:ReplicateImage` - ソースアカウントがイメージを送信先レジストリにレプリケートすることを許可します。
- `ecr:CreateRepository` - リポジトリがまだ存在しない場合、ECR が宛先レジストリにリポジトリを自動的に作成することを許可します。

> **Important**
>
> `ecr:CreateRepository` アクセス許可を付与しない場合は、レプリケーションが成功する前に、レプリケート先アカウントで同じ名前のリポジトリを手動で作成する必要があります。

送信先レジストリポリシーの例:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCrossAccountReplication",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": [
                "ecr:ReplicateImage",
                "ecr:CreateRepository"
            ],
            "Resource": "*"
        }
    ]
}
```

ソースアカウント要件

ソースアカウントには以下のことが必要です。

- レジストリ設定でレプリケーションルールを設定して、レプリケート先アカウントとリージョンを指定する。
- レプリケーションを設定する IAM プリンシパルに必要な ECR アクセス許可があることを確認する。

ソースリポジトリに追加のポリシーは必要ありません。ソースリポジトリには、レプリケーション許可を付与するリポジトリポリシーは必要ありません。

一般的な誤解

ECR クロスアカウントレプリケーションポリシーに関する一般的な誤解を以下に示します。

- 誤解: ソースリポジトリには、送信先アカウントがイメージをレプリケートできるようにするポリシーが必要です。

  現実: ソースリポジトリには、レプリケーションのための特別なポリシーは必要ありません。
- 誤解: 送信元アカウントと送信先アカウントの両方にレジストリポリシーが必要です。

  現実: レジストリアクセス許可ポリシーが必要なのは、送信先アカウントのみです。
- 誤解: リポジトリポリシーとレジストリポリシーは同じものです。

  現実: リポジトリポリシーは個々のリポジトリへのアクセスを制御し、レジストリポリシーはレプリケーションなどのレジストリレベルのオペレーションを制御します。

レプリケーション失敗のトラブルシューティング

クロスアカウントレプリケーションが失敗する場合は、以下を確認してください。

- 送信先アカウントにレジストリアクセス許可ポリシーが設定されていることを確認します。
- レジストリポリシーに `ecr:ReplicateImage` アクションと `ecr:CreateRepository` アクションの両方が含まれていることを確認します。
- 送信元アカウント ID が送信先レジストリポリシーで正しく指定されていることを確認します。

- 送信先リポジトリが存在することを確認する (ecr:CreateRepository が付与されていない場合)
- CloudTrail ログで失敗した CreateRepository コールまたは ReplicateImage API コールを確認する

プライベートイメージのレプリケーションに関する考慮事項

プライベートイメージのレプリケーションを使用する際には、以下の点を考慮する必要があります。

- レプリケーションの設定後にリポジトリにプッシュまたは復元されたリポジトリコンテンツのみがレプリケートされます。リポジトリ内の既存のコンテンツはいずれも複製されません。レプリケーションがオンになった後にイメージが復元されると、レプリケートされます。レプリケーションがオンになる前に復元された場合、レプリケートされません。
- レプリケーションが行われても、リポジトリ名はリージョンとアカウント間で同じままです。Amazon ECR はレプリケーション中のリポジトリ名の変更をサポートしていません。
- レプリケーション用にプライベートレジストリを初めて設定すると、Amazon ECR は、お客様に代わってサービスにリンクされた IAM ロールを作成します。サービスにリンクされた IAM ロールは、レジストリでレポジトリの作成とイメージのレプリケートを行うために必要なアクセス許可を Amazon ECR レプリケーションサービスに付与します。詳細については、「Amazon ECR でのサービスにリンクされたロールの使用」を参照してください。
- クロスアカウントレプリケーションを実行するには、プライベートレジストリのレプリケート先が、ソースレジストリによるそのイメージのレプリケートを許可する必要があります。これは、プライベートレジストリの許可ポリシーを設定することによって許可できます。詳細については、「Amazon ECR でのプライベートレジストリのアクセス許可」を参照してください。
- プライベートレジストリの許可ポリシーが許可を取り消すように変更された場合でも、以前に許可された進行中のレプリケーションは完了することができます。
- クロスリージョンレプリケーションを行うには、そのリージョン内またはリージョンに対してレプリケーションアクションが発生する前に、ソースアカウントとターゲットアカウントの両方がリージョンにオプトインされている必要があります。詳細については、「Amazon Web Services 全般のリファレンス」の「AWS リージョンの管理」を参照してください。
- AWS パーティション間のクロスリージョンレプリケーションはサポートされていません。たとえば、us-west-2 のリポジトリは cn-north-1 にレプリケートできません。 AWS パーティションの詳細については、AWS 「 全般のリファレンス」の「ARN 形式」を参照してください。
- プライベートレジストリのレプリケーション設定には、すべてのルール (合計で最大 10 個) 全体で最大 25 個の一意のレプリケート先を含めることができます。各ルールには、最大 100 個のフィル

ターを含めることができます。これは、例えば本番環境やテスト用に使用されるイメージが含まれるリポジトリに対して個別のルールを指定することを可能にします。

- レプリケーション設定は、リポジトリプレフィックスを指定することによって、プライベートレジストリ内のレプリケートされるリポジトリのフィルタリングをサポートします。例については、「例: リポジトリフィルターを使用したクロスリージョンレプリケーションの設定」を参照してください。
- レプリケーションアクションは、イメージのプッシュまたはイメージの復元ごとに 1 回のみ発生します。たとえば、us-west-2 から us-east-1 および us-east-1 から us-east-2 へのクロスリージョンレプリケーションを設定した場合、us-west-2 にプッシュされたイメージは us-east-1 にのみレプリケートされ、us-east-2 には再度レプリk-トされません。この動作は、クロスリージョンおよびクロスアカウントのレプリケーションの両方に適用されます。
- 大部分のイメージは 30 分以内にレプリケートされますが、まれにレプリケーションに時間がかかることがあります。
- レジストリレプリケーションでは、削除アクションやアーカイブアクションは実行されません。レプリケートされたイメージとリポジトリは、使用されなくなったときに削除またはアーカイブできます。
- レプリケートするイメージがレプリケート先でアーカイブされている場合、レプリケート先で復元されます。
- イメージがソースリージョンにアーカイブされている場合、レプリケーション設定で指定されたレプリケート先リージョンにはアーカイブされません。
- IAM ポリシーやライフサイクルポリシーなどのリポジトリポリシーはレプリケートされず、定義されているリポジトリ以外には影響しません。
- リポジトリ設定はデフォルトでレプリケートされません。リポジトリ作成テンプレートを使用してリポジトリ設定をレプリケートできます。これらの設定には、タグのミュータビリティ、暗号化、リポジトリのアクセス許可、ライフサイクルポリシーが含まれます。リポジトリ作成テンプレートの詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。
- リポジトリでタグのイミュータブル性が有効で、既存のイメージと同じタグを使用するイメージをレプリケートする場合、イメージはレプリケートされますが、重複したタグは含まれません。その結果、イメージのタグ付けが解除される可能性があります。
- イメージをレプリケートするときに、BLOB マウントが設定されている場合、ECR はソースリポジトリのレイヤーがレプリケート先レジストリに既に存在することを確認します。レプリケート先レジストリにレイヤーが既に存在する場合、ECR はそれらのレイヤーをマウントします。

> **Note**
>
> ソースレジストリがレプリケート先レジストリと異なる場合、ECR がレプリケートされたレイヤーをマウントするには、両方のレジストリで BLOB マウントを有効にする必要があります。

Amazon ECR でのプライベートイメージレプリケーションの例

以下の例は、プライベートのイメージレプリケーションの一般的な使用事例を示しています。を使用してレプリケーションを設定する場合は AWS CLI、JSON ファイルを作成する際の出発点として JSON の例を使用できます。 AWS マネジメントコンソールを使用してレプリケーションを設定する場合は、[確認して送信] ページでレプリケーションルールを確認すると、同様の JSON が表示されます。

例: 単一の送信先リージョンへのクロスリージョンレプリケーションの設定

次に、単一のレジストリ内でのクロスリージョンレプリケーション設定の例を示します。この例では、アカウント ID が 111122223333 であること、およびこのレプリケーション設定が us-west-2 以外のリージョンで指定されていることを前提とします。

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-2",
                    "registryId": "111122223333"
                }
            ]
        }
    ]
}
```

例: リポジトリフィルターを使用したクロスリージョンレプリケーションの設定

以下は、プレフィックス名の値と一致するリポジトリのクロスリージョンレプリケーションを設定する例です。この例は、アカウント ID が 111122223333 であること、us-west-1 以外のリージョ

ンでこのレプリケーション設定を指定していること、およびプレフィックスが prod のリポジトリがあることを前提としています。

```
{
 "rules": [{
  "destinations": [{
   "region": "us-west-1",
   "registryId": "111122223333"
  }],
  "repositoryFilters": [{
   "filter": "prod",
   "filterType": "PREFIX_MATCH"
  }]
 }]
}
```

例: 複数の送信先リージョンへのクロスリージョンレプリケーションの設定

次に、単一のレジストリ内でのクロスリージョンレプリケーション設定の例を示します。この例では、アカウント ID が 111122223333であり、 us-west-1または us-west-2 以外のリージョンでこのレプリケーション設定を指定することを前提としています。

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-1",
                    "registryId": "111122223333"
                },
                {
                    "region": "us-west-2",
                    "registryId": "111122223333"
                }
            ]
        }
    ]
}
```

例: クロスアカウントレプリケーションの設定

次に、レジストリのクロスアカウントレプリケーション設定の例を示します。この例では、444455556666 アカウントと us-west-2 リージョンも対するレプリケーションを設定します。

> **Important**
>
> クロスアカウントレプリケーションを実行するには、ターゲットアカウントで、レプリケーションを許可するレジストリのアクセス許可ポリシーを設定する必要があります。詳細については、「Amazon ECR でのプライベートレジストリのアクセス許可」を参照してください。

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-2",
                    "registryId": "444455556666"
                }
            ]
        }
    ]
}
```

例: 1 つの設定内での複数のルールの指定

以下は、レジストリに複数のレプリケーションルールを設定する例です。この例では、プレフィックスが のリポジトリを*############### ###################### 1 ######### ##111122223333* アカウントのレプリケーションを設定しますtestus-east-2。 prod us-west-2レプリケーション設定には最大 10 個のルールを含めることができ、各ルールが最大 25 個の送信先を指定します。

```
{
 "rules": [{
   "destinations": [{
    "region": "us-west-2",
    "registryId": "111122223333"
```

```
    }],
    "repositoryFilters": [{
     "filter": "prod",
     "filterType": "PREFIX_MATCH"
    }]
   },
   {
    "destinations": [{
     "region": "us-east-2",
     "registryId": "111122223333"
    }],
    "repositoryFilters": [{
     "filter": "test",
     "filterType": "PREFIX_MATCH"
    }]
   }
  ]
}
```

例: すべてのレプリケーション設定を削除する

レジストリからすべてのレプリケーション設定を削除する例を次に示します。レプリケーション設定を削除するには、空のルール配列を設定する必要があります。

```
{
    "rules": []
}
```

> **Important**
>
> レプリケーション設定を削除しても、以前にレプリケートされたリポジトリやイメージは削除されません。AMI が不要になった場合は、手動で削除する必要があります。

Amazon ECR でのプライベートイメージレプリケーションの設定

プライベートレジストリのリージョンごとのレプリケーションを設定します。クロスリージョンまたはクロスアカウントのレプリケーションを設定できます。

レプリケーションの一般的な使用方法の例については、Amazon ECR でのプライベートイメージレプリケーションの例 を参照してください。

レジストリのレプリケーション設定を構成するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、レジストリのレプリケーション設定を構成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ) を選択します。
4. プライベートレジストリページで、設定を選択し、レプリケーション設定で編集を選択します。
5. [Replication] (レプリケーション) ページで [Add replication rule] (レプリケーションルールを追加) をクリックします。
6. [Destination types] (送信先タイプ) ページで、クロスリージョンレプリケーションもしくはクロスアカウントレプリケーションのいずれか、またはそれら両方を有効にしてから、[Next] (次へ) をクリックします。
7. クロスリージョンレプリケーションを有効にした場合は、[Configure destination regions] (送信先リージョンを設定) で、1 つ、または複数の [Destination regions] (送信先リージョン) を選択し、[Next] (次へ) をクリックします。
8. クロスアカウントレプリケーションを有効にした場合は、[Cross-account replication] (クロスアカウントレプリケーション) で、レジストリのクロスアカウントレプリケーション設定を選択します。[Destination account] (送信先アカウント) には送信先アカウントのアカウント ID を入力し、レプリケート先にする 1 つ、または複数の [Destination regions] (送信先リージョン) を選択します。レプリケーションの送信先として追加のアカウントを設定するには、[Destination account +] (送信先アカウント +) をクリックします。

   > **Important**
   >
   > クロスアカウントレプリケーションを実行するには、ターゲットアカウントで、レプリケーションを許可するレジストリのアクセス許可ポリシーを設定する必要があります。詳細については、「Amazon ECR でのプライベートレジストリのアクセス許可」を参照してください。

9. (オプション) [Add filters] (フィルターを追加) ページで、レプリケーションルールに対して 1 つ、または複数のフィルターを指定してから、[Add] (追加) をクリックします。レプリケーションアクションに関連付けるフィルターごとに、このステップを繰り返します。フィルターはリポジトリ名のプレフィックスとして指定する必要があります。フィルターを追加しない場合、すべてのリポジトリの内容が複製されます。[Next] (次へ) をクリックすると、すべてのフィルターが追加されます。

10. [Review and submit] (確認して送信) ページでレプリケーションルールの設定を確認してから、[Submit rule] (ルールを送信) をクリックします。

レジストリのレプリケーション設定を構成するには (AWS CLI)

1. レジストリ用に定義するレプリケーションルールが含まれた JSON ファイルを作成します。レプリケーション設定には最大 10 個のルールを含めることができ、各ルールが最大 25 個の送信先と 100 個のフィルターを指定します。独自のアカウント内でクロスリージョンレプリケーションを設定するには、独自のアカウント ID を指定します。その他の例については、「Amazon ECR でのプライベートイメージレプリケーションの例」を参照してください。

```
{
 "rules": [{
  "destinations": [{
   "region": "destination_region",
   "registryId": "destination_accountId"
  }],
  "repositoryFilters": [{
   "filter": "repository_prefix_name",
   "filterType": "PREFIX_MATCH"
  }]
 }]
}
```

2. レジストリのレプリケーション設定を作成します。

```
aws ecr put-replication-configuration \
     --replication-configuration file://replication-settings.json \
     --region us-west-2
```

3. レジストリ設定を確認します。

```
aws ecr describe-registry \
     --region us-west-2
```

Amazon ECR でのプライベートイメージレプリケーション設定の削除

プライベートレジストリのレプリケーション設定を削除または無効化するには、空のレプリケーション設定を設定する必要があります。 AWS CLIに専用の削除コマンドはありません。

レジストリレプリケーション設定を削除するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、レジストリのレプリケーション設定を削除するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ) を選択します。
4. プライベートレジストリページで、設定を選択し、レプリケーション設定で編集を選択します。
5. 各ルールの削除オプションを選択して、既存のレプリケーションルールをすべて削除します。
6. [保存] を選択して、空のレプリケーション設定を適用します。

レジストリレプリケーション設定を削除するには (AWS CLI)

1. 空のルール配列を持つ JSON ファイルを作成して、すべてのレプリケーション設定を削除します。

```
{
    "rules": []
}
```

2. 空のレプリケーション設定をレジストリに適用します。

```
aws ecr put-replication-configuration \
     --replication-configuration file://empty-replication-settings.json \
     --region us-west-2
```

3. レプリケーション設定が削除されていることを確認します。

```
aws ecr describe-registry \
     --region us-west-2
```

出力には、ルールのない空の `replicationConfiguration` が表示されます。

> **Important**
>
> レプリケーション設定を削除しても、以前にレプリケートされたリポジトリやイメージは削除されません。AMI が不要になった場合は、手動で削除する必要があります。

プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート

Amazon ECR リポジトリ作成テンプレートを使用して、Amazon ECR がユーザーに代わって作成するリポジトリ用の設定を定義します。リポジトリ作成テンプレートの設定はリポジトリの作成時にのみ適用され、既存のリポジトリや他の方法で作成されたリポジトリには影響しません。現在、リポジトリ作成テンプレートを使用すると、リポジトリの作成時に以下の機能に対して設定を適用できます。

- プルスルーキャッシュ
- プッシュ時に作成する
- レプリケーション

リポジトリ作成テンプレートの仕組み

Amazon ECR がユーザーに代わって新しいプライベートリポジトリを作成する必要がある場合があります。例えば、次のようになります。

- プルスルーキャッシュルールを初めて使用して、アップストリームリポジトリのコンテンツを取得し、Amazon ECR プライベートレジストリに保存する場合。
- イメージがまだ存在しないリポジトリにプッシュする場合。
- Amazon ECR にリポジトリを別のリージョンまたはアカウントにレプリケートさせる場合。

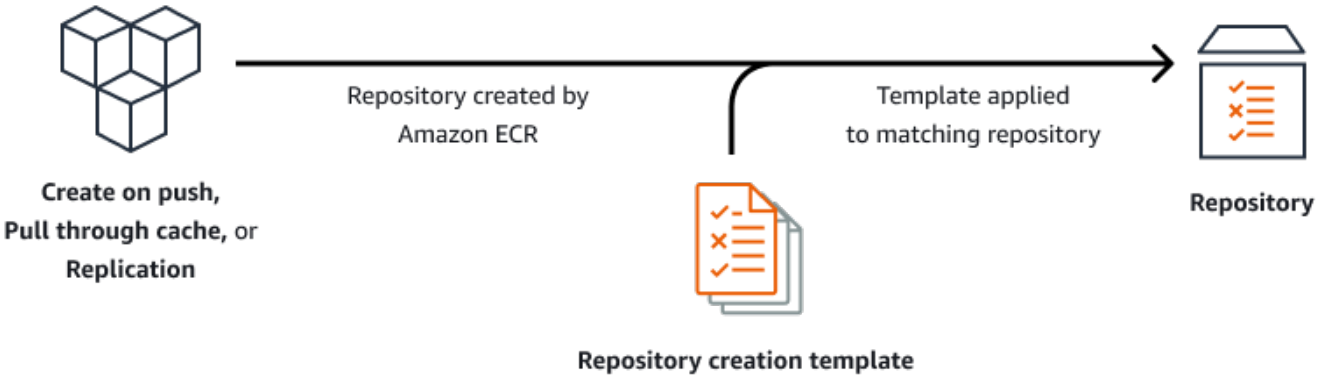
プルスルーキャッシュルールまたはレプリケートされたリポジトリに一致するリポジトリ作成テンプレートがない場合、Amazon ECR は新しいリポジトリのデフォルト設定を使用します。これらのデフォルト設定には、タグの不変性をオフにする、AES-256 暗号化を使用する、リポジトリやライフサイクルポリシーを一切適用しないなどが含まれます。

イメージプッシュのターゲットリポジトリに一致するリポジトリ作成テンプレートがない場合、Amazon ECR はデフォルト設定でリポジトリを作成しません。

リポジトリ作成テンプレートを使用すると、Amazon ECR がプルスルーキャッシュ、プッシュ時作成、レプリケーションアクションを通じて作成された新しいリポジトリに適用する設定を定義でき

ます。新しいリポジトリのタグ不変性、暗号化設定、リポジトリ権限、ライフサイクルポリシー、リソースタグを定義できます。

次の図は、リポジトリ作成テンプレートを使用するときに Amazon ECR が使用するワークフローを示しています。



以下では、リポジトリ作成テンプレートの各パラメータについて詳しく説明します。

Prefix

[プレフィックス] は、テンプレートに関連付けるリポジトリ名前空間のプレフィックスです。このプレフィックスを使用して作成されたすべてのリポジトリには、このテンプレートで定義されている設定が適用されます。例えば、`prod` というプレフィックスは、`prod/` で始まるすべてのリポジトリに適用されます。同様に、`prod/team` というプレフィックスは、`prod/team/` で始まるすべてのリポジトリに適用されます。2 つのテンプレートを含むレジストリでは、1 つのテンプレートにプレフィックス「prod」が付いており、もう 1 つのテンプレートにプレフィックス「prod/team」が付いている場合、名前が「prod/team/」で始まるすべてのリポジトリに、プレフィックス「prod/team」が付いているテンプレートが適用されます。

作成テンプレートが関連付けられていないレジストリ内のすべてのリポジトリにテンプレートを適用するには、プレフィックスとして `ROOT` を使用できます。

> **Important**
>
> プレフィックスの末尾には常に `/` が適用されると想定されます。`ecr-public` をプレフィックスとして指定すると、Amazon ECR はそれを `ecr-public/` として扱います。プルスルーキャッシュルールを使用する場合、ルール作成時に指定するリポジトリプレフィックスは、リポジトリ作成テンプレートのプレフィックスとしても指定する必要があります。

説明

この [テンプレートの説明] はオプションで、リポジトリ作成テンプレートの目的を説明するために使用されます。

適用対象

[適用対象] 設定は、このテンプレートを使用して作成する Amazon ECR 作成リポジトリを決定します。有効な値は PULL_THROUGH_CACHE、CREATE_ON_PUSH、および REPLICATION です。例えば、プルスルーキャッシュルールを初めて使用して、アップストリームリポジトリのコンテンツを取得し、Amazon ECR プライベートレジストリに保存します。プルスルーキャッシュルールに一致するリポジトリ作成テンプレートがない場合、Amazon ECR は新しいリポジトリのデフォルト設定を使用します。

リポジトリ作成ロール

[リポジトリ作成ロール] は、リポジトリ作成テンプレートを使用してリポジトリを作成および設定する際に、Amazon ECR が引き受ける IAM ロール ARN です。このロールは、テンプレートでリポジトリタグや KMS を使用する場合に指定する必要があります。指定しないと、リポジトリの作成は失敗します。

イメージタグの変更可能性

テンプレートを使用して作成されたリポジトリに使用するタグの変更可能性設定です。このパラメータを省略すると、MUTABLE のデフォルト設定が使用されます。デフォルト設定では、イメージタグの上書きが許可されます。これは、プルスルーキャッシュアクションによって作成されたリポジトリに使用されるテンプレートに使用することを推奨する設定です。これにより、タグが同じ場合でも Amazon ECR はキャッシュされたイメージを更新できます。

IMMUTABLE を指定すると、リポジトリ内のすべてのイメージタグは不変となり、上書きが禁止されます。

暗号化設定

> **Important**
>
> AWS KMS (DSSE-KMS) を使用した二層式サーバー側の暗号化は、 AWS GovCloud (US) リージョンでのみ使用できます。

テンプレートを使用して作成されたリポジトリに使用する暗号化設定です。

KMS 暗号化タイプを使用する場合、リポジトリのコンテンツは、 AWS KMSに保存されている AWS Key Management Service キーにより、サーバー側の暗号化を使用して暗号化されます。AWS KMS を使用してデータを暗号化する場合、Amazon ECR のデフォルトの AWS マネージド AWS KMS キーを使用するか、既に作成した独自の AWS KMS キーを指定できます。さらに、で単層または二層暗号化を使用するように選択できます AWS KMS。詳細については、「保管中の暗号化」を参照してください。KMS 暗号化タイプを使用していて、それをクロスリージョンレプリケーションで使用する場合は、追加のアクセス許可が必要になることがあります。詳細については、「Creating a KMS key policy for replication」を参照してください。

AES256 の暗号化タイプを使用する場合、Amazon ECR は Amazon S3 で管理された暗号化キーによりサーバー側の暗号化キーを使用して、AES-256 暗号化アルゴリズムを使用するリポジトリ内のイメージを暗号化します。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 マネージドキーによるサーバー側の暗号化 (SSE-S3)」を参照してください。

リポジトリ権限

テンプレートを使用して作成されたリポジトリに適用するリポジトリポリシーです。リポジトリポリシーは、リソースベースのアクセス権限を使用してリポジトリへのアクセスを制御します。リソースベースのアクセス権限により、どの IAM ユーザーあるいはロールがリポジトリにアクセスでき、どのようなアクションを実行できるかを指定できます。デフォルトでは、リポジトリを作成した AWS アカウントのみがリポジトリにアクセスできます。リポジトリへの追加のアクセス許可を付与または拒否するポリシードキュメントを適用できます。詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。

リポジトリライフサイクルポリシー

テンプレートを使用して作成されたリポジトリに使用するライフサイクルポリシーです。ライフサイクルポリシーを使用すると、プライベートリポジトリ内のイメージのライフサイクル管理をより詳細に制御できます。ライフサイクルポリシーは 1 つまたは複数のルールで、各ルールでは Amazon ECR へのアクションが定義されています。時間やカウント数に基づいてイメージの有効期限を設定することによってコンテナイメージを自動的にクリーンアップできます。詳細については、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。

リソースタグ

リソースタグは、リポジトリに適用し、分類と整理に役立つメタデータです。各タグはキーとオプションの値で構成され、どちらもお客様側が定義します。クロスリージョンレプリケーションでリポジトリ作成テンプレートを使用する場合は、このアクセス許可を宛先レジストリポリシーに適用する必要があります。

Amazon ECR でのリポジトリ作成テンプレートの作成

リポジトリ作成テンプレートを作成して、プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に Amazon ECR がユーザーに代わって作成したリポジトリに使用する設定を定義できます。リポジトリ作成テンプレートが作成されると、作成されたすべての新しいリポジトリにその設定が適用されます。これは以前に作成されたリポジトリには影響しません。

テンプレートを使用してリポジトリを設定する際に、KMS キーとリソースタグを指定するオプションがあります。1 つ以上のテンプレートで KMS キー、リソースタグ、またはその両方の組み合わせを使用する場合は、以下を行う必要があります。

- リポジトリ作成テンプレートのカスタムポリシーの作成.
- リポジトリ作成テンプレート用の IAM ロールの作成.

設定が完了したら、カスタムロールをレジストリ内の特定のテンプレートにアタッチできます。

リポジトリ作成テンプレートを作成するための IAM アクセス許可

IAM プリンシパルがリポジトリ作成テンプレートを管理するには、以下のアクセス許可が必要です。これらのアクセス許可は、アイデンティティに基づく IAM ポリシーを使用して付与する必要があります。

- `ecr:CreateRepositoryCreationTemplate` — リポジトリ作成テンプレートを作成するアクセス許可を付与します。
- `ecr:UpdateRepositoryCreationTemplate` — リポジトリ作成テンプレートを更新するためのアクセス許可を付与します。
- `ecr:DescribeRepositoryCreationTemplates` – レジストリ内のリポジトリ作成テンプレートをリストするためのアクセス許可を付与します。
- `ecr:DeleteRepositoryCreationTemplate` — リポジトリ作成テンプレートを削除するアクセス許可を付与します。
- `ecr:CreateRepository` – Amazon ECR リポジトリを作成するためのアクセス許可を付与します。
- `ecr:PutLifecyclePolicy` — ライフサイクルポリシーを作成し、それをリポジトリに適用するアクセス許可を付与します。このアクセス許可は、リポジトリ作成テンプレートにライフサイクルポリシーが含まれている場合にのみ必要です。

- `ecr:SetRepositoryPolicy` — リポジトリのアクセス許可ポリシーを作成するアクセス許可を付与します。このアクセス許可は、リポジトリ作成テンプレートにリポジトリポリシーが含まれている場合にのみ必要です。
- `iam:PassRole` – エンティティがサービスまたはアプリケーションにロールを渡すことを許可するためのアクセス許可を付与します。このアクセス許可は、ユーザーに代わってアクションを実行するロールを引き受ける必要があるサービスやアプリケーションに必要です。

リポジトリ作成テンプレートのカスタムポリシーの作成

を使用して、後で IAM ロールに関連付けられるポリシー AWS マネジメントコンソール を定義できます。その後、この IAM ロールは、リポジトリ作成テンプレートを設定するときにリポジトリ作成ロールとして使用できます。

AWS マネジメントコンソール

JSON ポリシーエディタを使用してリポジトリ作成テンプレートのカスタムポリシーを作成するには、次の手順を実行します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
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] セクションで、[JSON] オプションを選択します。
5. [JSON] フィールドに次のポリシーを入力します。

JSON

```
{
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "ecr:CreateRepository",
                    "ecr:ReplicateImage",
                    "ecr:TagResource"
                ],
                "Resource": "*"
```

```
            },
            {
                "Effect": "Allow",
                "Action": [
                    "kms:CreateGrant",
                    "kms:RetireGrant",
                    "kms:DescribeKey"
                ],
                "Resource": "*"
            }
        ]
}
```

6. ポリシーの検証中に生成されたセキュリティ警告、エラー、または一般警告をすべて解決してから、[次へ] を選択します。
7. ポリシーにアクセス権限を追加し終えたら、[次へ] を選択します。
8. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。
9. ポリシーを作成 をクリックして、新しいポリシーを保存します。
10. 作成テンプレート用にこのポリシーを割り当てるロールを作成します。「リポジトリ作成テンプレート用の IAM ロールの作成」を参照してください。

リポジトリ作成テンプレート用の IAM ロールの作成

を使用して、テンプレートでリポジトリタグまたは KMS を使用しているリポジトリ作成テンプレートでリポジトリ作成ロールを指定するときに、Amazon ECR で使用できるロール AWS マネジメントコンソール を作成できます。

AWS マネジメントコンソール

ロールを作成するには、次の手順を実行します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. コンソールのナビゲーションペインで、[ロール]、[ロールの作成] の順に選択します。
3. [カスタム信頼ポリシー] ロールタイプを選択します。

4. [カスタム信頼ポリシー] セクションで、以下にリストされているカスタム信頼ポリシーを貼り付けます。

   JSON

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

5. [次へ] を選択します。
6. [アクセス許可の追加] ページで、アクセス許可ポリシーのリストから、先ほど作成したカスタムポリシーの横にあるチェックボックスを選択し、[次へ] を選択します。
7. ロール名 に、ロールの名前を入力します。ロール名は 内で一意である必要があります AWS アカウント。ロール名がポリシーとして、または ARN の一部として使用される場合、ロール名は大文字と小文字を区別します。サインイン処理中など、コンソールでロール名がユーザーに表示される場合、ロール名は大文字と小文字を区別しません。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、作成後にロール名を編集することはできません。
8. (オプション) [説明] には、新しいロールの説明を入力します。
9. ロール情報を確認し、ロールの作成 を選択します。

リポジトリ作成テンプレートの作成

テンプレートに必要な前提条件をすべて満たしたら、リポジトリ作成テンプレートの作成に進むことができます。

AWS マネジメントコンソール

リポジトリ作成テンプレートを作成するには (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、リポジトリ作成テンプレートを作成するリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ]、[リポジトリ作成テンプレート] を選択します。
4. [リポジトリ作成テンプレート] ページで、[テンプレートを作成] を選択します。
5. [ステップ 1: テンプレートの定義] ページの [テンプレートの詳細] で、[特定のプレフィックス] を選択してテンプレートを特定のリポジトリ名前空間のプレフィックスに適用するか、[ECR レジストリの任意のプレフィックス] を選択して、リージョン内の他のテンプレートと一致しないすべてのリポジトリにテンプレートを適用します。

   a. [特定のプレフィックス] を選択した場合は、[プレフィックス] にテンプレートを適用するリポジトリ名前空間プレフィックスを指定します。プレフィックスの末尾には常に / が適用されると想定されます。例えば、`prod` というプレフィックスは、`prod/` で始まるすべてのリポジトリに適用されます。同様に、`prod/team` というプレフィックスは、`prod/team/` で始まるすべてのリポジトリに適用されます。

   b. [ECR レジストリの任意のプレフィックス] を選択すると、[プレフィックス] は `ROOT` に設定されます。

6. [適用対象] では、このテンプレートを適用する Amazon ECR ワークフローを指定します。使用できるオプションは `PULL_THROUGH_CACHE`、`CREATE_ON_PUSH`、および `REPLICATION` です。
7. [テンプレートの説明] には、テンプレートの説明をオプションで指定し、[次へ] を選択します。
8. [ステップ 2: リポジトリ作成設定の追加] ページで、テンプレートを使用して作成されたリポジトリに適用するリポジトリ設定設定を指定します。

   a. [イメージタグのミュータビリティ] で、このリポジトリのタグの変更可能性の設定を選択します。詳細については、「Amazon ECR でのイメージタグの上書きの防止」を参照してください。

      - [ミュータブル] – イメージタグを上書きする場合は、このオプションを選択します。Amazon ECR がキャッシュされたイメージを更新できるように、プルスルー

キャッシュアクションを使用するリポジトリに推奨されます。さらに、いくつかのミュータブルタグのタグ更新を無効にするには、[Mutable tag exclusion] テキストボックスにタグ名を入力します。複数の類似したタグが一致するワイルドカード (*) を使用できます。

- [イミュータブル] – イメージタグが上書きされるのを防ぎ、既存のタグを含むイメージをプッシュするときに、リポジトリ内のすべてのタグと除外に適用する場合、このオプションを選択します。Amazon ECR は、既存のタグを使用してイメージをプッシュしようとすると `ImageTagAlreadyExistsException` を返します。さらに、いくつかのイミュータブルタグのタグ更新を有効にするには、[Immutable tag exclusion] テキストボックスにタグ名を入力します。複数の類似したタグが一致するワイルドカード (*) を使用できます。

b. [暗号化設定] では、使用する暗号化設定を選択します。詳細については、「保管中の暗号化」を参照してください。

[AES-256] が選択されていると、Amazon ECR は、Amazon Simple Storage Service が管理する暗号化キーによるサーバー側暗号化を使用し、保管中のデータが業界標準の AES-256 暗号化アルゴリズムで暗号化されます。これは追加コストなしで提供されます。

[AWS KMS] を選択すると、Amazon ECR は AWS Key Management Service (AWS KMS) に保存されたキーによるサーバー側の暗号化を使用します。 AWS KMS を使用してデータを暗号化する場合、Amazon ECR によって管理されるデフォルトの AWS マネージドキーを使用するか、カスタマーマネージドキーと呼ばれる独自の AWS KMS キーを指定できます。

> **Note**
>
> 一度リポジトリを作成すると、リポジトリの暗号化設定は変更できません。

c. [リポジトリ権限] については、このテンプレートを使用して作成されたリポジトリに適用するリポジトリ権限ポリシーを指定します。オプションで、ドロップダウンを使用して、最も一般的なユースケース用の JSON サンプルを 1 つ選択できます。詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。

d. [リポジトリライフサイクルポリシー] では、このテンプレートを使用して作成されたリポジトリに適用するリポジトリライフサイクルポリシーを指定します。オプションで、ドロップダウンを使用して、最も一般的なユースケース用の JSON サンプルを 1 つ選択

できます。詳細については、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。

e. リポジトリ AWS タグでは、キーと値のペアの形式でメタデータを指定して、このテンプレートを使用して作成されたリポジトリに関連付け、次へを選択します。詳細については、「Amazon ECR でのプライベートリポジトリのタグ付け」を参照してください。

f. [リポジトリ作成ロール] では、テンプレートでリポジトリタグまたは KMS を使用する場合にリポジトリ作成テンプレートに使用するカスタム IAM ロールをドロップダウンメニューから選択します (詳細については「リポジトリ作成テンプレート用の IAM ロールの作成」を参照)。その後、[次へ] を選択します。

9. [ステップ 3: 確認と作成] ページで、リポジトリ作成テンプレートに指定した設定を確認します。[編集] を選択して、変更を加えます。完了したら、[作成] を選択します。

AWS CLI

create-repository-creation-template AWS CLI コマンドは、プライベートレジストリのリポジトリ作成テンプレートを作成するために使用されます。

リポジトリ作成テンプレートを作成するには (AWS CLI)

1. を使用して、create-repository-creation-template コマンドのスケルトン AWS CLI を生成します。

```
aws ecr create-repository-creation-template \
    --generate-cli-skeleton
```

コマンドの出力には、リポジトリ作成テンプレートの完全な構文が表示されます。

```
{
"appliedFor":[""], // string array, but valid are PULL_THROUGH_CACHE,
 CREATE_ON_PUSH, and REPLICATION
"prefix": "string",
    "description": "string",
    "imageTagMutability":
 "MUTABLE"|"IMMUTABLE"|"IMMUTABLE_WITH_EXCLUSION"|"MUTABLE_WITH_EXCLUSION",
    "imageTagMutabilityExclusionFilters": [
        "filterType": "WILDCARD",
        "filter": "string"
    ],
    "repositoryPolicy": "string",
```

```
    "lifecyclePolicy": "string"
"encryptionConfiguration": {
"encryptionType": "AES256"|"KMS",
        "kmsKey": "string"
    },
    "resourceTags": [
        {
"Key": "string",
            "Value": "string"
        }
    ],
    "customRoleArn": "string", // must be a valid IAM Role ARN
}
```

2. 前のステップの出力を使用して、repository-creation-template.json という名前のファイルを作成します。このテンプレートは、将来のリポジトリへのイメージのプッシュとプルを許可するリポジトリポリシーを使用して prod/* の下に作成されたリポジトリに対する KMS 暗号化キーを設定し、2 週間以上経過したイメージを期限切れにするライフサイクルポリシーを設定し、ECR が KMS キーにアクセスしてリソースタグ examplekey を将来のリポジトリに割り当てることができるようにするカスタムロールを設定します。

```
{
"prefix": "prod",
    "description": "For repositories cached from my PTC rule and in my
 replication configuration that start with 'prod/'",
    "appliedFor": ["PULL_THROUGH_CACHE", "CREATE_ON_PUSH", "REPLICATION"],
    "encryptionConfiguration": {
"encryptionType": "KMS",
        "kmsKey": "arn:aws:kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-
cdef-example11111"
    },
    "resourceTags": [
        {
"Key": "examplekey",
            "Value": "examplevalue"
        }
    ],
    "imageTagMutability": "IMMUTABLE_WITH_EXCLUSION",
    "imageTagMutabilityExclusionFilters": [
      {
      "filterType": "WILDCARD",
      "filter": "latest"
      },
```

```
      {
      "filterType": "WILDCARD",
      "filter": "beta*"
      }
    ]
    "repositoryPolicy": "{\"Version\":\"2012-10-17\",       \"Statement\":
[{\"Sid\":\"AllowPushPullIAMRole\",\"Effect\":\"Allow\",\"Principal\":{\"AWS\":
\"arn:aws:iam::111122223333:user\/IAMusername\"},\"Action\":[\"ecr:BatchGetImage
\",\"ecr:BatchCheckLayerAvailability\",\"ecr:CompleteLayerUpload\",
\"ecr:GetDownloadUrlForLayer\",\"ecr:InitiateLayerUpload\",\"ecr:PutImage\",
\"ecr:UploadLayerPart\"]}]}",
    "lifecyclePolicy": "{\"rules\":[{\"rulePriority\":1,\"description\":\"Expire
 images older than 14 days\",\"selection\":{\"tagStatus\":\"any\",\"countType
\":\"sinceImagePushed\",\"countUnit\":\"days\",\"countNumber\":14},\"action\":
{\"type\":\"expire\"}}]}",
    "customRoleArn": "arn:aws:iam::111122223333:role/myRole"
}
```

3. 次のコマンドを使用して、リポジトリ作成テンプレートを作成します。次の例では、repository-creation-template.json の代わりに、前のステップで作成した設定ファイルの名前を指定することを確認してください。

```
aws ecr create-repository-creation-template \
    --cli-input-json file://repository-creation-template.json
```

リポジトリ作成テンプレートを更新する

リポジトリ作成テンプレートは、設定を変更する必要がある場合には編集することができます。リポジトリ作成テンプレートを編集すると、新しい設定が既存のテンプレートに適用されます。

> **Important**
>
> これは以前に作成されたリポジトリには影響しません。

AWS マネジメントコンソール

リポジトリ作成テンプレートを編集する手順 (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。

2. ナビゲーションバーから、編集するリポジトリ作成テンプレートがあるリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ] を選択し、次に [スキャン] を選択します。
4. ナビゲーションバーから、[リポジトリ作成テンプレート] を選択します。
5. [リポジトリ作成テンプレート] ページで、編集するリポジトリ作成テンプレートを選択します。
6. [アクション] ドロップダウンメニューから、[編集] を選択します。
7. 設定を確認して更新します。
8. [更新] を選択して、新しい作成テンプレートの設定を適用します。

AWS CLI

リポジトリ作成テンプレートを編集する手順 (AWS CLI)

- update-repository-creation-template コマンドを使用して、既存のリポジトリ作成テンプレートを更新します。テンプレートの `prefix` 値を指定する必要があります。次の例では、`prod` プレフィックスを持つリポジトリ作成テンプレートを更新します。

```
aws ecr update-repository-creation-template \
     --prefix prod \
     --image-tag-mutability="IMMUTABLE_WITH_EXCLUSION" \
     --image-tag-mutability-exclusion-filters filterType=WILDCARD, filter=latest
```

コマンドの出力には、更新されたリポジトリ作成テンプレートの詳細が表示されます。

Amazon ECR でのリポジトリ作成テンプレートの削除

不要になったリポジトリ作成テンプレートを削除できます。リポジトリ作成テンプレートを削除すると、プルスルーキャッシュまたはレプリケーションのアクション中に関連付けられたプレフィックスの下に新しく作成されたリポジトリは、一致する別のテンプレートが見つからない限り、デフォルト設定を継承します。「リポジトリ作成テンプレートの仕組み」を参照してください。

> **Important**
>
> これは以前に作成されたリポジトリには影響しません。

AWS マネジメントコンソール

リポジトリ作成テンプレートを削除するには (AWS マネジメントコンソール)

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、削除するリポジトリ作成テンプレートがあるリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ]、[リポジトリ作成テンプレート] を選択します。
4. [リポジトリ作成テンプレート] ページで、削除するリポジトリ作成テンプレートを選択します。
5. [アクション] ドロップダウンメニューから [削除] を選択します。

AWS CLI

リポジトリ作成テンプレートを削除するには (AWS CLI)

- delete-repository-creation-template.html コマンドを使用して、既存のリポジトリ作成テンプレートを削除します。テンプレートの `prefix` 値を指定する必要があります。次の例では、`prod` プレフィックスを持つリポジトリ作成テンプレートを削除します。

```
aws ecr delete-repository-creation-template \
     --prefix prod
```

コマンドの出力には、削除されたリポジトリ作成テンプレートの詳細が表示されます。

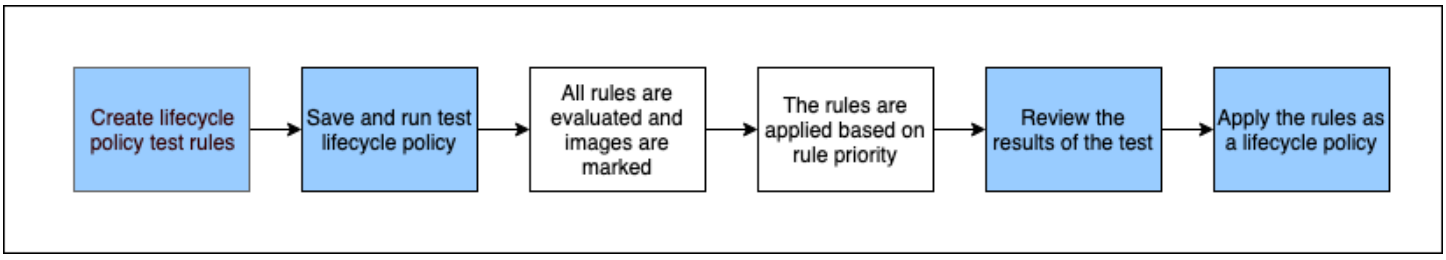
Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化

Amazon ECR ライフサイクルポリシーを使用すると、プライベートリポジトリ内のイメージのライフサイクル管理をより詳細に制御できます。ライフサイクルポリシーには 1 つ以上のルールが含まれ、各ルールで Amazon ECR のアクションが定義されます。ライフサイクルポリシーの有効期限基準に基づいて、ライフサイクルポリシーで指定された基準に基づいて 24 時間以内にイメージをアーカイブまたは期限切れにすることができます。Amazon ECR がライフサイクルポリシーに基づいてアクションを実行した場合、このアクションはイベントとして AWS CloudTrailにキャプチャされます。詳細については、「を使用した Amazon ECR アクションの口グ記録 AWS CloudTrail」を参照してください。

ライフサイクルポリシーの機能

ライフサイクルポリシーは、リポジトリ内のどのイメージに有効期限切れにするかを決定する 1 つ以上のルールで構成されています。ライフサイクルポリシーの使用を検討する場合、リポジトリに適用する前に、ライフサイクルポリシーのプレビューでライフサイクルポリシーが有効期限切れになったイメージを確認することが重要です。ライフサイクルポリシーがリポジトリに適用されると、イメージが有効期限の基準を満たしてから 24 時間以内に期限切れになることが予想されます。Amazon ECR がライフサイクルポリシーに基づいてアクションを実行すると、これはイベントとして AWS CloudTrailにキャプチャされます。詳細については、「を使用した Amazon ECR アクションの口グ記録 AWS CloudTrail」を参照してください。

次の図表は、ライフサイクルポリシーのワークフローを示します。



1. テストルールを 1 つ以上作成します。
2. テストルールを保存し、プレビューを実行します。
3. ライフサイクルポリシーエバリュエーターは、すべてのルールを評価し、各ルールが影響するイメージにマークを付けます。

4. 次に、ライフサイクルポリシーエバリュエーターはルールの優先度に基づいてルールを適用し、リポジトリ内のどのイメージの有効期限が切れるかアーカイブされるかを表示します。ルールの優先度の数値が低いほど、優先度が高くなります。例えば、優先度 1 のルールは、優先度 2 のルールよりも優先されます。
5. テストの結果を確認し、期限切れまたはアーカイブとマークされたイメージが意図したとおりであることを確認します。
6. テストルールをリポジトリのライフサイクルポリシーとして適用します。
7. ライフサイクルポリシーが作成されると、イメージの有効期限が切れるか、有効期限の基準を満たしてから 24 時間以内にアーカイブされるはずです。

ライフサイクルポリシーの評価ルール

ライフサイクルポリシーエバリュエーターは、ライフサイクルポリシーのプレーンテキスト JSON を解析して、すべてのルールを評価し、ルールの優先順位に基づいてリポジトリ内のイメージに適用します。次に、ライフサイクルポリシーエバリュエーターのロジックについて詳しく説明します。例については「Amazon ECR でのライフサイクルポリシーの例」を参照してください。

- リポジトリに参照アーティファクトが存在する場合、Amazon ECR ライフサイクルポリシーは、対象イメージの削除またはアーカイブから 24 時間以内にそれらのアーティファクトを自動的に期限切れまたはアーカイブします。
- 優先順位に関係なく、すべてのルールが同時に評価されます。すべてのルールが評価された後、優先度に基づいてルールが適用されます。
- イメージの有効期限が切れているか、1 つまたはゼロのルールによってアーカイブされています。
- ルールのタグ付け要件に一致するイメージを、優先度の低いルールで期限切れにしたりアーカイブしたりすることはできません。
- ルールは、優先度の高いルールでマークされたイメージをマークすることはできませんが、有効期限が切れていないかアーカイブされていないかのように識別できます。
- 特定のストレージクラスを選択するすべてのルールのセットには、一意のプレフィックスのセットが含まれている必要があります。
- タグ付けされていないイメージを選択できるのは、特定のストレージクラスを選択する 1 つのルールのみです。
- イメージがマニフェストリストによって参照されている場合、マニフェストリストが最初に削除またはアーカイブされない限り、有効期限が切れたりアーカイブされたりすることはできません。

- 有効期限は常に pushed_at_timeまたは によって順序付けられtransitioned_at_time、古いイメージは新しいイメージより前に期限切れになります。イメージがアーカイブされ、過去の任意の時点で復元された場合、イメージの last_activated_atが の代わりに使用されますpushed_at_time。
- ライフサイクルポリシールールでは、tagPatternList または tagPrefixList のいずれかを指定できますが、両方は指定できません。ただし、ライフサイクルポリシーにはパターンリストとプレフィックスリストの両方を使用する複数のルールが含まれる場合があります。tagPatternList または tagPrefixList の値に含まれるすべてのタグがイメージのいずれかのタグと一致した場合に、イメージは一致したことになります。
- tagPatternList または tagPrefixList パラメータは、tagStatus が tagged の場合にのみ使用できます。
- tagPatternList を使用する場合、ワイルドカードフィルターに一致すればイメージがマークされます。例えば、prod* のフィルターを適用すると、prod で始まる名前 (prod、prod1、production-team1 など) を持つリポジトリと一致します。同様に、*prod* のフィルターを適用すると、名前に prod が含まれる (repo-production、prod-team など) リポジトリと一致します。

  > **Important**
  >
  > 1 文字列あたりのワイルドカード (*) の上限は 4 つです。例えば、["test*1*2*3*4*5*6"] は有効ですが ["*test*1*2*3", "test*1*2*3*"] は無効です。

- tagPrefixList を使用した場合、tagPrefixList 値のすべてのワイルドカードフィルターがイメージのいずれかのタグと一致した場合に、イメージは一致したことになります。
- countUnit パラメータは、 countTypeが sinceImagePushed、sinceImagePulled、またはの場合にのみ使用されますsinceImageTransitioned。
- ではcountType = imageCountMoreThan、イメージは に基づいて最も新しいイメージから古いイメージにソートされpushed_at_time、指定された数を超えるすべてのイメージは期限切れまたはアーカイブされます。
- ではcountType = sinceImagePushed、 に基づく指定された日数よりもpushed_at_time古いすべてのイメージcountNumberが期限切れまたはアーカイブされます。
- ではcountType = sinceImagePulled、 に基づく指定された日数よりも古いすべてのイメージlast_recorded_pulltimeがcountNumberアーカイブされます。イメージがプルされたことがない場合は、 の代わりにイメージの pushed_at_timeが使用されま

すlast_recorded_pulltime。イメージがアーカイブされてから復元されたが、復元されてからプルされなかった場合、イメージの last_activated_atは の代わりに使用されますlast_recorded_pulltime。

- ではcountType = sinceImageTransitioned、 に基づく指定された日数よりもlast_archived_at古いアーカイブ済みイメージはすべて期限切れcountNumberになります。
- 有効期限は常に によって順序付けpushed_at_timeされ、古いイメージは新しいイメージより前に有効期限が切れます。

Amazon ECR でのライフサイクルポリシーのプレビューの作成

ライフサイクルポリシーのプレビューを使用して、ライフサイクルポリシーを適用する前にイメージリポジトリへの影響を確認できます。ベストプラクティスとして、リポジトリにライフサイクルポリシーを適用する前に、プレビューを実行することをお勧めします。

> **Note**
>
> Amazon ECR レプリケーションを使用して異なるリージョンまたはアカウントにまたがるリポジトリのコピーを作成する場合、ライフサイクルポリシーは、作成されたリージョンのリポジトリに対してのみアクションを実行可能であることに注意してください。したがって、レプリケーションをオンにしている場合は、リポジトリを複製する各リージョンとアカウントにライフサイクルポリシーを作成してください。

ライフサイクルポリシーのプレビューを作成するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、ライフサイクルポリシーのプレビューを実行するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[プライベートレジストリ] の下で、[リポジトリ] を選択します。
4. [プライベートリポジトリ] ページでリポジトリを選択し、[アクション] ドロップダウンを使用して [ライフサイクルポリシー] を選択します。
5. リポジトリのライフサイクルポリシーページで、[テストルールの編集]、[ルールの作成] の順に選択します。
6. ライフサイクルポリシーのルールに以下の詳細を指定します。

a. [ルールの優先順位] で、ルールの優先順位を数字で入力します。ルールの優先順位によって、ライフサイクルポリシールールが適用される順序が決まります。数値が小さいほど優先順位が高くなります。例えば、優先度 1 のルールは、優先度 2 のルールよりも優先されます。

b. [ルールの説明] で、ライフサイクルポリシーのルールの説明を入力します。

c. [イメージのステータス] では、[タグ付き (ワイルドカードマッチング)]、[タグ付き (プレフィックスマッチング)]、[タグ付けなし]、または [任意] を選択します。

> **Important**
>
> 複数のタグを指定する場合、指定されたすべてのタグが付いているイメージのみが選択されます。

d. [イメージのステータス] で [タグ付き (ワイルドカードマッチング)] を選択した場合、[ワイルドカードマッチング用のタグを指定] では、ライフサイクルポリシーに基づいてアクションを実行するためのワイルドカード (*) 付きのイメージタグのリストを指定できます。例えば、イメージに `prod`、`prod1`、`prod2` というようにタグが付いている場合、すべてのアクションを実行する `prod*` を指定します。複数のタグを指定する場合、指定されたすべてのタグが付いているイメージのみが選択されます。

> **Important**
>
> 1 文字列あたりのワイルドカード (*) の上限は 4 つです。例えば、`["test*1*2*3*4*5*6"]` は有効ですが `["*test*1*2*3", "test*1*2*3*"]` は無効です。

e. [イメージのステータス] で [タグ付き (プレフィックスマッチング)] を選択した場合、[ワイルドカードマッチング用のタグを指定] では、ライフサイクルポリシーに基づいてアクションを実行するためのイメージタグのリストを指定できます。

f. 一致基準では、イメージ作成からの日数、最後に記録されたプル時間からの日数、イメージアーカイブからの日数、またはイメージ数を選択し、値を指定します。

g. ルールアクションで、期限切れまたはアーカイブを選択します。

h. [保存] を選択します。

7. ステップ 5 ~ 7 を繰り返すことにより、追加のライフサイクルポリシーのルールを作成します。

8. ライフサイクルポリシーのプレビューを実行するには、[テストの保存と実行] を選択します。
9. [イメージがテストライフサイクルルールに一致しました] で、ライフサイクルポリシーのプレビューの影響を確認します。
10. プレビューの結果に満足したら、[Apply as lifecycle policy] を選択して指定したルールでライフサイクルポリシーを作成します。ライフサイクルポリシーを適用した後、影響を受けるイメージの有効期限が切れるか、24 時間以内にアーカイブされるはずです。
11. プレビュー結果に満足できない場合は、1 つ以上のテストライフサイクルルールを削除し、1 つ以上のルールを作成して置き換え、テストを繰り返すことができます。

Amazon ECR でのリポジトリのライフサイクルポリシーの作成

ライフサイクルポリシーを使用して、未使用のリポジトリイメージを期限切れまたはアーカイブする一連のルールを作成します。ライフサイクルポリシーを作成すると、影響を受けるイメージの有効期限が切れるか、24 時間以内にアーカイブされます。

> **Note**
>
> Amazon ECR レプリケーションを使用して異なるリージョンまたはアカウントにまたがるリポジトリのコピーを作成する場合、ライフサイクルポリシーは、作成されたリージョンのリポジトリに対してのみアクションを実行可能であることに注意してください。したがって、レプリケーションをオンにしている場合は、リポジトリを複製する各リージョンとアカウントにライフサイクルポリシーを作成してください。

前提条件

ベストプラクティス: ライフサイクルポリシープレビューを作成して、ライフサイクルポリシールールによって期限切れまたはアーカイブされたイメージが意図したとおりであることを確認します。手順については、「Amazon ECR でのライフサイクルポリシーのプレビューの作成」を参照してください。

ライフサイクルポリシーを作成するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/repositories で Amazon ECR コンソールを開きます。
2. ナビゲーションバーから、ライフサイクルポリシーを作成するリポジトリを含むリージョンを選択します。

3. ナビゲーションペインで、[プライベートレジストリ] の下で、[リポジトリ] を選択します。
4. [プライベートリポジトリ] ページでリポジトリを選択し、[アクション] ドロップダウンを使用して [ライフサイクルポリシー] を選択します。
5. リポジトリのライフサイクルポリシーで、[ルールを作成] を選択します。
6. ライフサイクルポリシーのルールに以下の詳細を入力します。

   a. [ルールの優先順位] で、ルールの優先順位を数字で入力します。ルールの優先順位によって、ライフサイクルポリシールールが適用される順序が決まります。ルールの優先度の数値が低いほど、優先度が高くなります。例えば、優先度 1 のルールは、優先度 2 のルールよりも優先されます。

   b. [ルールの説明] で、ライフサイクルポリシーのルールの説明を入力します。

   c. [イメージのステータス] では、[タグ付き (ワイルドカードマッチング)]、[タグ付き (プレフィックスマッチング)]、[タグ付けなし]、または [任意] を選択します。

   > **Important**
   >
   > 複数のタグを指定する場合、指定されたすべてのタグが付いているイメージのみが選択されます。

   d. [イメージのステータス] で [タグ付き (ワイルドカードマッチング)] を選択した場合、[ワイルドカードマッチング用のタグを指定] では、ライフサイクルポリシーに基づいてアクションを実行するためのワイルドカード (*) 付きのイメージタグのリストを指定できます。例えば、イメージに `prod`、`prod1`、`prod2` というようにタグが付いている場合、すべてのアクションを実行する `prod*` を指定します。複数のタグを指定する場合、指定されたすべてのタグが付いているイメージのみが選択されます。

   > **Important**
   >
   > 1 文字列あたりのワイルドカード (*) の上限は 4 つです。例えば、`["test*1*2*3*4*5*6"]` は有効ですが `["*test*1*2*3", "test*1*2*3*"]` は無効です。

   e. [イメージのステータス] で [タグ付き (プレフィックスマッチング)] を選択した場合、[ワイルドカードマッチング用のタグを指定] では、ライフサイクルポリシーに基づいてアクションを実行するためのイメージタグのリストを指定できます。

f. 一致基準では、イメージ作成からの日数、最後に記録されたプル時間からの日数、イメージアーカイブからの日数、またはイメージ数を選択し、値を指定します。

g. ルールアクションで、期限切れまたはアーカイブを選択します。

h. [保存] を選択します。

7. ステップ 5 ~ 7 を繰り返すことにより、追加のライフサイクルポリシーのルールを作成します。

ライフサイクルポリシーを作成するには (AWS CLI)

1. ライフサイクルポリシーを作成するリポジトリの名前を取得します。

```
aws ecr describe-repositories
```

2. ライフサイクルポリシーの内容を指定して、policy.json というローカルファイル名を作成します。ライフサイクルポリシーの例については、「Amazon ECR でのライフサイクルポリシーの例」を参照してください。

3. リポジトリ名を指定してライフサイクルポリシーを作成し、作成したライフサイクルポリシー JSON ファイルを参照します。

```
aws ecr put-lifecycle-policy \
      --repository-name repository-name \
      --lifecycle-policy-text file://policy.json
```

Amazon ECR でのライフサイクルポリシーの例

以下は、ライフサイクルポリシーの構文を示した例です。

ポリシープロパティの詳細については、「Amazon ECR でのライフサイクルポリシーのプロパティ」を参照してください。を使用してライフサイクルポリシーを作成する手順については AWS CLI、「」を参照してくださいライフサイクルポリシーを作成するには (AWS CLI)。

ライフサイクルポリシーのテンプレート

ライフサイクルポリシーのコンテンツは、リポジトリに関連付けられる前に評価されます。以下に示しているのは、ライフサイクルポリシーの JSON 構文テンプレートです。

```
{
```

```
        "rules": [
            {
                "rulePriority": integer,
                "description": "string",
                "selection": {
                    "tagStatus": "tagged"|"untagged"|"any",
                    "tagPatternList": list<string>,
                    "tagPrefixList": list<string>,
                    "storageClass": "standard"|"archive",
                    "countType":
 "imageCountMoreThan"|"sinceImagePushed"|"sinceImagePulled"|"sinceImageTransitioned",
                    "countUnit": "string",
                    "countNumber": integer
                },
                "action": {
                    "type": "expire"|"transition",
                    "targetStorageClass": "archive"
                }
            }
        ]
    }
```

イメージの経過日数によるフィルタリング

次の例は、prod* の tagPatternList が同様に 14 日より古いものを使用することにより、prod で始まるタグでイメージを有効期限切れにするポリシーのライフサイクルポリシー構文を示しています。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Expire images older than 14 days",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["prod*"],
                "countType": "sinceImagePushed",
                "countUnit": "days",
                "countNumber": 14
            },
            "action": {
                "type": "expire"
            }
```

```
        }
    ]
}
```

最終プル時刻のフィルタリング

次の例は、イメージを数90日でプルされていないアーカイブストレージに移行するポリシーのライフサイクルポリシー構文を示しています。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Archive images not pulled in 90 days",
            "selection": {
                "tagStatus": "any",
                "countType": "sinceImagePulled",
                "countUnit": "days",
                "countNumber": 90
            },
            "action": {
                "type": "transition",
                "targetStorageClass": "archive"
            }
        }
    ]
}
```

Important

sinceImagePulled カウントタイプは transitionアクションで使用する必要があります。expire アクションでは使用できません。プルアクティビティに基づいてイメージを削除するには、まず を使用してアーカイブストレージに移行してからsinceImagePulled、expireアクションsinceImageTransitionedで を使用してイメージを削除します。イメージは、削除前に少なくとも 90 日間アーカイブストレージに保存する必要があります。

アーカイブ移行時間のフィルタリング

次の例は、アーカイブストレージに 365日以上保存されているアーカイブイメージの有効期限が切れるポリシーのライフサイクルポリシー構文を示しています。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Expire images archived for more than 365 days",
            "selection": {
                "tagStatus": "any",
                "storageClass": "archive",
                "countType": "sinceImageTransitioned",
                "countUnit": "days",
                "countNumber": 365
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

Important

`sinceImageTransitioned` カウントタイプは、 `expire`アクションと`archive`ストレージクラスで使用する必要があります。イメージは、削除前に少なくとも 90 日間アーカイブストレージに保存する必要があります。

イメージ数によるフィルタリング

次の例で、タグ付けされていないイメージを 1 つだけ保持して残りはすべて期限切れにするポリシーのライフサイクルポリシーの構文を示します。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Keep only one untagged image, expire all others",
```

```
            "selection": {
                "tagStatus": "untagged",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

複数のルールによるフィルタリング

以下は、ライフサイクルポリシーで複数のルールを使用する例です。サンプルのリポジトリとライフサイクルポリシーが結果の説明とともに示されています。

例 A

リポジトリのコンテンツ

- Image A, Taglist: ["beta-1", "prod-1"], Pushed: 10 days ago
- Image B, Taglist: ["beta-2", "prod-2"], Pushed: 9 days ago
- Image C, Taglist: ["beta-3"], Pushed: 8 days ago

ライフサイクルポリシーのテキスト

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["prod*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
```

```
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["beta*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、prod というプレフィックスでタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。イメージ A が期限切れとしてマークされます。
- ルール 2 は、beta というプレフィックスでタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。イメージ A とイメージ B の両方が期限切れとしてマークされます。ただし、イメージ A はルール 1 ですでに確認されていて、もしイメージ B が期限切れであれば、ルール 1 に違反してしまうので、スキップされます。
- 結果: イメージ A は期限切れです。

例 B

これは前の例と同じリポジトリですが、結果を説明するためにルールの優先順位が変更されています。

リポジトリのコンテンツ

- Image A, Taglist: ["beta-1", "prod-1"], Pushed: 10 days ago
- Image B, Taglist: ["beta-2", "prod-2"], Pushed: 9 days ago
- Image C, Taglist: ["beta-3"], Pushed: 8 days ago

ライフサイクルポリシーのテキスト

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["beta*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["prod*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、beta というプレフィックスでタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。3 つのすべてのイメージが確認され、イメージ A とイメージ B が期限切れとしてマークされます。
- ルール 2 は、prod というプレフィックスでタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。今回は、使用可能なイメージはすべてルール 1 で確認済みのため、確認できるイメージがありません。そのため、追加でイメージをマークすることはありません。

- 結果: イメージ A と B は期限切れです。

単一のルールで複数のタグをフィルタリングする

次の例で、1 つのルールでの複数のタグパターンのライフサイクルポリシーの構文を指定します。サンプルのリポジトリとライフサイクルポリシーが結果の説明とともに示されています。

例 A

1 つのルールで複数のタグパターンが指定されたときは、イメージはすべてのリストされたタグパターンに一致する必要があります。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-1"], Pushed: 12 days ago
- Image B, Taglist: ["beta-1"], Pushed: 11 days ago
- Image C, Taglist: ["alpha-2", "beta-2"], Pushed: 10 days ago
- Image D, Taglist: ["alpha-3"], Pushed: 4 days ago
- Image E, Taglist: ["beta-3"], Pushed: 3 days ago
- Image F, Taglist: ["alpha-4", "beta-4"], Pushed: 2 days ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["alpha*", "beta*"],
                "countType": "sinceImagePushed",
                "countNumber": 5,
                "countUnit": "days"
            },
            "action": {
                "type": "expire"
            }
        }
    ]
```

```
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、alpha と beta というプレフィックスでタグ付けされたイメージを特定します。イメージ C と F が確認されます。5 日より古いイメージをマークするので、イメージ C がマークされます。
- 結果: イメージ C は期限切れです。

例 B

次の例では、タグは排他的ではないことを説明します。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-1", "beta-1", "gamma-1"], Pushed: 10 days ago
- Image B, Taglist: ["alpha-2", "beta-2"], Pushed: 9 days ago
- Image C, Taglist: ["alpha-3", "beta-3", "gamma-2"], Pushed: 8 days ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["alpha*", "beta*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、alpha と beta というプレフィックスでタグ付けされたイメージを特定します。すべてのイメージが確認されます。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。イメージ A と B が期限切れとしてマークされます。
- 結果: イメージ A と B は期限切れです。

すべてのイメージでのフィルタリング

次のライフサイクルポリシーの例では、異なるフィルタですべてのイメージを指定します。サンプルのリポジトリとライフサイクルポリシーが結果の説明とともに示されています。

例 A

次に、すべてのルールを適用する一方、イメージを 1 つだけ保持して残りはすべて期限切れにするポリシーのライフサイクルポリシーの構文を示します。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-1"], Pushed: 4 days ago
- Image B, Taglist: ["beta-1"], Pushed: 3 days ago
- Image C, Taglist: [], Pushed: 2 days ago
- Image D, Taglist: ["alpha-2"], Pushed: 1 day ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、すべてのイメージを特定します。イメージ A、B、C、D が確認されます。最も新しいもの以外のすべてのイメージが期限切れとされます。イメージ A、B、C が期限切れとしてマークされます。
- 結果: イメージ A、B、C は期限切れです。

例 B

以下の例は、単一のポリシーのすべてのルールのタイプを組み合わせるライフサイクルポリシーを示しています。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-", "beta-1", "-1"], Pushed: 4 days ago
- Image B, Taglist: [], Pushed: 3 days ago
- Image C, Taglist: ["alpha-2"], Pushed: 2 days ago
- Image D, Taglist: ["git hash"], Pushed: 1 day ago
- Image E, Taglist: [], Pushed: 1 day ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["alpha*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
```

```
                "tagStatus": "untagged",
                "countType": "sinceImagePushed",
                "countUnit": "days",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 3,
            "description": "Rule 3",
            "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーの口ジックは次のようになります。

- ルール 1 は、alpha というプレフィックスでタグ付けされたイメージを特定します。イメージ A と C を識別します。最も新しいイメージを保持し、残りは期限切れとしてマークされます。イメージ A が期限切れとしてマークされます。
- ルール 2 は、タグ付けされていないイメージを特定します。イメージ B と E を識別します。有効期限が 1 日より古いすべてのイメージがマークされます。イメージ B が期限切れとしてマークされます。
- ルール 3 は、すべてのイメージを特定します。イメージ A、B、C、D、E を識別します。最も新しいイメージを保持し、残りは期限切れとしてマークされます。ただし、優先度がより高いルールにより識別されるため、イメージ A、B、C、または E をマークすることはできません。イメージ D が期限切れとしてマークされます。
- 結果: イメージ A、B、D は期限切れです。

アーカイブの例

次の例は、イメージを削除する代わりにアーカイブするライフサイクルポリシーを示しています。

指定した日数より古いイメージのアーカイブ

次の例は、30 日以上経過した で始まるタグを持つイメージprodをアーカイブするライフサイクルポリシーを示しています。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Archive production images older than 30 days",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["prod*"],
                "countType": "sinceImagePushed",
                "countUnit": "days",
                "countNumber": 30
            },
            "action": {
                "type": "transition",
                "targetStorageClass": "archive"
            }
        }
    ]
}
```

指定された日数内にプルされないイメージのアーカイブ

次の例は、90 日以内にプルされていないイメージをアーカイブするライフサイクルポリシーを示しています。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Archive images not pulled in 90 days",
            "selection": {
                "tagStatus": "any",
```

```
                "countType": "sinceImagePulled",
                "countUnit": "days",
                "countNumber": 90
            },
            "action": {
                "type": "transition",
                "targetStorageClass": "archive"
            }
        }
    ]
}
```

アーカイブルールと有効期限ルールの組み合わせ

次の例は、30 日以上経過したイメージをアーカイブし、365 日以上アーカイブされたイメージを完全に期限切れにするライフサイクルポリシーを示しています。

> **Note**
>
> アーカイブされたイメージの最小ストレージ期間は 90 日間です。アーカイブに 90 日未満のイメージを削除するライフサイクルポリシーを設定することはできません。アーカイブ期間が 90 日未満のイメージを削除する必要がある場合は、 batch-delete-image API を使用する必要がありますが、90 日間の最小ストレージ期間に対して課金されます。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Archive images older than 30 days",
            "selection": {
                "tagStatus": "any",
                "countType": "sinceImagePushed",
                "countUnit": "days",
                "countNumber": 30
            },
            "action": {
                "type": "transition",
                "targetStorageClass": "archive"
            }
        },
```

```
        {
            "rulePriority": 2,
            "description": "Expire images archived for more than 365 days",
            "selection": {
                "tagStatus": "any",
                "storageClass": "archive",
                "countType": "sinceImageTransitioned",
                "countUnit": "days",
                "countNumber": 365
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

Amazon ECR でのライフサイクルポリシーのプロパティ

ライフサイクルポリシーには次のプロパティがあります。

ライフサイクルポリシーの例については、「Amazon ECR でのライフサイクルポリシーの例」を参照してください。を使用してライフサイクルポリシーを作成する手順については AWS CLI、「」を参照してくださいライフサイクルポリシーを作成するには (AWS CLI)。

ルールの優先順位

`rulePriority`

タイプ: 整数

必須: はい

ルールを適用する順序を低いものから高いものの順に設定します。優先順位が 1 のライフサイクルポリシールールが最初に適用され、次に優先順位 2 のルールが適用され、その後も同様に続きます。ライフサイクルポリシーにルールを追加するときは、それぞれに `rulePriority` の一意の値を付ける必要があります。ポリシー内のルール間で値が連続している必要はありません。any の `tagStatus` を持つルールは、`rulePriority` の最大値を持ち、最後に評価される必要があります。

説明

`description`

型: 文字列

必須: いいえ

(オプション) ライフサイクルポリシー内のルールの目的について説明します。

タグステータス

`tagStatus`

型: 文字列

必須: はい

追加するライフサイクルポリシーのルールがイメージのタグを指定するかどうかを決定します。使用できるオプションは、tagged、untagged、または any です。any を指定する場合は、すべてのイメージに対してルールが評価されます。を指定する場合はtagged、 tagPrefixList 値または tagPatternList値も指定する必要があります。を指定する場合はuntagged、tagPrefixListと の両方を省略する必要がありますtagPatternList。

タグパターンリスト

`tagPatternList`

タイプ: list[string]

必須:はい、tagPrefixList がタグ付きに設定されていて、tagStatus が指定されていない場合

タグ付きイメージのライフサイクルポリシーを作成するときは、tagPatternList を使用してタグの有効期限を指定するのがベストプラクティスです。ライフサイクルポリシーでアクションを実行するときワイルドカード (*) を含む可能性のあるイメージタグパターンのカンマ区切りリストを指定する必要があります。例えば、イメージに prod、prod1、prod2 というようにタグが付いている場合、すべてを指定するためにタグパターンリスト prod* を使用します。複数のタグを指定する場合、指定されたすべてのタグが付いているイメージのみが選択されます。

> **Important**
>
> 1 文字列あたりのワイルドカード (*) の上限は 4 つです。例えば、["test*1*2*3*4*5*6"] は有効ですが ["*test*1*2*3", "test*1*2*3*"] は無効です。

タグプレフィックスリスト

`tagPrefixList`

タイプ: list[string]

必須:はい、tagPatternList がタグ付きに設定されていて、tagStatus が指定されていない場合

"tagStatus": "tagged" を指定し、tagPatternList を指定していない場合にのみ使用されます。ライフサイクルポリシーでアクションを実行するための、カンマ区切りのイメージタグプレフィックスのリストを指定する必要があります。たとえば、イメージに prod、prod1、prod2 というようにタグが付いている場合、すべてを指定するためにタグプレフィックス prod を使用します。複数のタグを指定する場合、指定されたすべてのタグが付いているイメージのみが選択されます。

ストレージクラス

`storageClass`

タイプ: 文字列

必須: はい、 countType が の場合 sinceImageTransitioned

ルールは、このストレージクラスのイメージのみを選択します。imageCountMoreThan、、sinceImagePushedまたは countTypeの を使用する場合sinceImagePulled、サポートされている値は のみですstandard。のカウントタイプを使用する場合sinceImageTransitioned、これは必須であり、サポートされている値は のみですarchive。これを省略すると、 の値standardが使用されます。

カウントタイプ

countType

型: 文字列

必須: はい

イメージに適用するカウントタイプを指定します。

countType が imageCountMoreThan に設定してある場合は、countNumber も指定して、リポジトリに存在するイメージ数の制限を設定するルールを作成します。countType が sinceImagePushed、、sinceImagePulledまたは に設定されている場合sinceImageTransitioned、リポジトリに存在するイメージの時間制限countNumberを指定するには、 countUnitと も指定します。

カウント単位

countUnit

タイプ: 文字列

必須: はい、 countTypeが sinceImagePushed、、sinceImagePulledまたは に設定されている場合のみ sinceImageTransitioned

日数を表す countNumber に加えて、カウント単位の days も時間単位として指定します。

これは、 countTypeが sinceImagePushed、sinceImagePulled、または の場合にのみ指定する必要がありますsinceImageTransitioned。 countTypeが他の値である場合、カウント単位を指定すると エラーが発生します。

カウント数

countNumber

タイプ: 整数

必須: はい

カウント数を指定します。許容値は正の整数です (0 は許容値ではありません)。

使用している countType が imageCountMoreThan である場合、この値はリポジトリに維持するイメージの最大数です。使用している countType が sinceImagePushed である場合、この値はイメージの最大期限です。countType 使用される が の場合sinceImagePulled、値はイメージが最後にプルされてからの最大日数です。countType 使用される が の場合sinceImageTransitioned、値はイメージがアーカイブされてからの最大日数です。

Action

type

型: 文字列

必須: はい

アクションタイプを指定します。サポートされている値は、 expire (イメージを削除) と transition (イメージをアーカイブストレージに移動する) です。

targetStorageClass

タイプ: 文字列

必須: はい、 type が の場合 transition

ライフサイクルポリシーでイメージを移行するストレージクラス。 archiveはサポートされている唯一の値です。

プルタイム更新の除外

Amazon ECR は、 AWS Inspector によるプルを除くすべてのプルの`LastRecordedPullTime`タイムスタンプを更新します。プルタイム更新の除外により、サードパーティーのスキャナー (Crowdstrike、Snyk、Trivy など) によるプルなど、イメージをプルするときにイメージのプル時間を更新しない IAM ロール ARNs を指定できます。これは、プル時間がライフサイクルポリシーの決定に影響を与えないようにするテストまたは CI/CD 目的で使用されるイメージに役立ちます。

除外リストのロールがイメージをプルしても、プル時間は変更されません。他のロールはプル時間 (現在の動作) を更新し続けます。アカウントごとに最大 100 個の除外を設定できます。

プルタイム更新の除外の管理

プルタイム更新の除外を管理するには、次の IAM アクセス許可が必要です。

- `ecr:CreatePullTimeUpdateExclusion` – 除外リストにロール ARN を追加するアクセス許可を付与します。
- `ecr:DeletePullTimeUpdateExclusion` – 除外リストからロール ARN を削除するアクセス許可を付与します。
- `ecr:ListPullTimeUpdateExclusions` – 除外リスト内のすべてのロール ARNs を一覧表示するアクセス許可を付与します。

> **Note**
>
> アクセス`iam:PassRole`許可は必要ありません。Amazon ECR はアクションを実行するロールを引き受けません。除外設定 ARNs のみを使用して、イメージのプル時間を更新する必要があるかどうかを判断します。

Amazon ECR コンソールまたは AWS CLI を使用して、プルタイム更新の除外を管理できます。

AWS マネジメントコンソール

プルタイム更新の除外を管理するには (AWS マネジメントコンソール)

1. https://console.aws.amazon.com/ecr/private-registry/repositories で Amazon ECR コンソールを開きます。

2. ナビゲーションバーからリージョンを選択します。
3. ナビゲーションペインで、プライベートレジストリ、機能と設定を選択し、プルタイム更新の除外を選択します。
4. 除外を追加するには、「除外の追加」を選択し、ロール ARN を入力して、「追加」を選択します。
5. 除外を削除するには、リストからロール ARN を選択し、削除を選択します。
6. すべての除外を表示するには、リストに設定されたすべてのロール ARNs が表示されます。

AWS CLI

プルタイム更新の除外を作成するには

- create-pull-time-update-exclusion コマンドを使用して、除外リストにロール ARN を追加します。

```
aws ecr create-pull-time-update-exclusion \
    --role-arn arn:aws:iam::123456789012:role/scanner-role
```

コマンドは、ロール ARN と作成タイムスタンプを返します。

```
{
   "roleArn": "arn:aws:iam::123456789012:role/scanner-role",
   "createdAt": 1745531331.0
}
```

プルタイム更新の除外を削除するには

- delete-pull-time-update-exclusion コマンドを使用して、除外リストからロール ARN を削除します。

```
aws ecr delete-pull-time-update-exclusion \
    --role-arn arn:aws:iam::123456789012:role/scanner-role
```

コマンドは、削除されたロール ARN を返します。

```
{
  "roleArn": "arn:aws:iam::123456789012:role/scanner-role"
```

```
}
```

プルタイム更新の除外を一覧表示するには

1. list-pull-time-update-exclusions コマンドを使用して、除外リスト内のすべてのロール ARNs を一覧表示します。

```
aws ecr list-pull-time-update-exclusions
```

除外が設定されていない場合、コマンドは空のリストを返します。

```
{
   "pullTimeUpdateExclusions": []
}
```

除外が設定されている場合、 コマンドはロール ARNs のリストを返します。

```
{
   "pullTimeUpdateExclusions": [
        "arn:aws:iam::123456789012:role/security-role"
    ]
}
```

2. 結果をページ分割するには、 --max-results および --next-tokenパラメータを使用します。

```
aws ecr list-pull-time-update-exclusions \
    --max-results 4
```

コマンドは、指定された数の結果まで返し、nextTokenさらに結果がある場合は を返します。

```
{
   "pullTimeUpdateExclusions": [
        "arn:aws:iam::123456789012:role/security-role1",
        "arn:aws:iam::123456789012:role/security-role2",
        "arn:aws:iam::123456789012:role/security-role3",
        "arn:aws:iam::123456789012:role/security-role4"
    ],
```

```
    "nextToken": "ukD72mdD/mC8b5xV3susmJzzaTgp3hKwR9nRUW1yZZ79..."
}
```

結果の次のページを取得するには、前のレスポンスnextTokenの を使用します。

```
aws ecr list-pull-time-update-exclusions \
    --max-results 4 \
    --next-token ukD72mdD/mC8b5xV3susmJzzaTgp3hKwR9nRUW1yZZ79...
```

プルタイム更新の除外に関する考慮事項

プルタイム更新の除外を使用する場合は、次の点を考慮してください。

- 除外を一覧表示するためのデフォルトのページサイズは 100 です。ページ分割は、maxResultsおよび nextTokenパラメータで使用できます。
- 正しい ARNs 形式の有効な IAM ロール ARN のみが受け入れられます。
- 既に存在する除外を作成しようとすると、ExclusionAlreadyExistsExceptionエラーが発生します。存在しない除外を削除しようとすると、ExclusionNotFoundExceptionエラーが発生します。

Amazon Elastic Container Registry 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 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、 AWS お客様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して説明しています。

- クラウドのセキュリティ – クラウドで AWS 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。「AWS 」 コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon ECR に適用されるコンプライアンスプログラムについては、「コンプライアンスプログラムによるAWS 対象範囲内のサービス」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、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon ECR 使用時における責任共有モデルの適用法を理解するのに役立ちます。以下のトピックで、セキュリティおよびコンプライアンスの目的を満たすように、Amazon ECR を設定する方法について説明します。また、Amazon ECR 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。

トピック

- Amazon Elastic Container Registry の Identity and Access Management
- Amazon ECR でのデータ保護
- Amazon Elastic Container Registry のコンプライアンス検証
- Amazon Elastic Container Registry のインフラストラクチャセキュリティ
- サービス間での不分別な代理処理の防止

Amazon Elastic Container Registry の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインインを許可) し、誰に

Amazon ECR リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- オーディエンス
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon Elastic Container Registry と IAM が連動するしくみ
- Amazon Elastic Container Registry のアイデンティティベースのポリシーの例
- タグベースのアクセスコントロールを使用する
- AWS Amazon Elastic Container Registry の マネージドポリシー
- Amazon ECR でのサービスにリンクされたロールの使用
- Amazon Elastic Container Registry の アイデンティティとアクセスのトラブルシューティング

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「Amazon Elastic Container Registry の アイデンティティとアクセスのトラブルシューティング」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「Amazon Elastic Container Registry と IAM が連動するしくみ」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「Amazon Elastic Container Registry のアイデンティティベースのポリシーの例」を参照)

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスすることを人間のユーザーに要求する AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられたときにアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリ

シーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。

- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

Amazon Elastic Container Registry と IAM が連動するしくみ

IAM を使用して Amazon ECR へのアクセスを管理する前に、Amazon ECR で使用できる IAM 機能について理解しておく必要があります。Amazon ECR およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

トピック

- Amazon ECR のアイデンティティベースのポリシー
- Amazon ECR のリソースベースのポリシー
- Amazon ECR タグに基づく認可
- Amazon ECR のIAM ロール

Amazon ECR のアイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは許可または拒否するアクションとリソース、またアクションを許可または拒否する条件を指定できます。Amazon ECR は、特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については、「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。

アクション

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられた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ポリシーで使用されます。

Amazon ECR のポリシーアクションは、アクションの前にプレフィックス `ecr:` を使用します。たとえば、Amazon ECR `CreateRepository` API オペレーションを使用して Amazon ECR リポジトリを作成するアクセス許可を付与するには、ポリシーに `ecr:CreateRepository` アクションを含めます。ポリシーステートメントには、`Action` または `NotAction` エレメントを含める必要があります。Amazon ECR は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": [
      "ecr:action1",
      "ecr:action2"
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、`Describe` という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "ecr:Describe*"
```

Amazon ECR アクションのリストについては、IAM ユーザーガイドの「Amazon Elastic Container Registry のアクション、リソース、および条件キー」を参照してください。

リソース

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Resource` JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベ

ルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Amazon ECR リポジトリリソースには、次の ARN があります。

```
arn:${Partition}:ecr:${Region}:${Account}:repository/${Repository-name}
```

ARN の形式の詳細については、「Amazon リソースネーム (ARNs AWS 「サービス名前空間」を参照してください。

たとえば、ステートメントの us-east-1 リージョン で my-repo リポジトリを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
```

特定のアカウントに属するすべての リポジトリを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/*"
```

複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": [
      "resource1",
      "resource2"
```

Amazon ECR リソースタイプとその ARN のリストを表示するには、IAM ユーザーガイドの「Amazon Elastic Container Registry で定義されるリソースタイプ」を参照してください。各リソースの ARN を指定できるアクションについては、「Amazon Elastic Container Registry で定義されるアクション」を参照してください。

条件キー

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」の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。

Amazon ECR では独自の条件キーが定義されており、また一部のグローバル条件キーの使用がサポートされています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドの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。

ほとんどの Amazon ECR アクションは、aws:ResourceTag と ecr:ResourceTag の条件キーをサポートします。詳細については、「タグベースのアクセスコントロールを使用する」を参照してください。

Amazon ECR 条件キーのリストについては、IAM ユーザーガイドの「Amazon Elastic Container Registry の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon Elastic Container Registry で定義されるアクション」を参照してください。

例

Amazon ECR でのアイデンティティベースのポリシーの例は、「Amazon Elastic Container Registry のアイデンティティベースのポリシーの例」を参照してください。

Amazon ECR のリソースベースのポリシー

リソースベースのポリシーとは、Amazon ECR リソース上で指定するプリンシパルとしてのどのアクションをどの条件で実行できるかを指定する JSON ポリシードキュメントです。Amazon ECR は、Amazon ECR リポジトリのリソースベースのアクセス許可ポリシーをサポートします。リソースベースのポリシーでは、リソースごとに他の アカウントに使用許可を付与することができます。リソースベースのポリシーを使用して、Amazon ECR リポジトリへのアクセスを AWS サービスに許可することもできます。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる AWS アカウントにある場合は、プリンシパルエンティティにリソースへのアクセス許可も付与する必要があります。アクセス許可は、アイデンティティベースのポリシーをエンティティにアタッチすることで付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーには追加の Amazon EcR リポジトリアクセス許可は必要ありません。詳細につい

ては、IAM ユーザーガイドの「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon ECR サービスでは、リポジトリポリシーと呼ばれるリソースベースのポリシーのタイプを 1 つのみサポートしており、これがリポジトリにアタッチされます。このポリシーは、リポジトリに対してアクションを実行できるプリンシパルエンティティ (アカウント、ユーザー、ロール、フェデレーティッドユーザー) を定義します。リソースベースのポリシーをリポジトリにアタッチする方法については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。

> **Note**
>
> Amazon ECR リポジトリポリシーでは、ポリシー要素 `Sid` は IAM ポリシーではサポートされていない追加の文字とスペースをサポートします。

例

Amazon ECR リソースベースのポリシーの例を表示するには、「Amazon ECR でのプライベートリポジトリポリシーの例」を参照してください。

Amazon ECR タグに基づく認可

タグは、Amazon ECR リソースにアタッチするか、Amazon ECR へのリクエストで渡すことができます。タグに基づいてアクセスを管理するには、`ecr:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`*、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。Amazon ECR リソースのタグ付けの詳細については、「Amazon ECR でのプライベートリポジトリのタグ付け」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグベースのアクセスコントロールを使用する」を参照してください。

Amazon ECR のIAM ロール

IAM ロールは、特定のアクセス許可を持つ AWS アカウント内のエンティティです。

Amazon ECR での一時的な認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインインする、IAM 役割を引き受ける、またはクロスアカウント役割を引き受けることができます。一時的なセキュリティ認証情報を取得するには、AssumeRole や GetFederationToken などの AWS STS API オペレーションを呼び出します。

Amazon ECR では、一時認証情報が使用できます。

サービスリンクロール

サービスにリンクされたロールを使用すると、 AWS サービスは他の サービスのリソースにアクセスして、ユーザーに代わってアクションを実行できます。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

Amazon ECR は、サービスにリンクされたロールをサポートしています。詳細については、「Amazon ECR でのサービスにリンクされたロールの使用」を参照してください。

Amazon Elastic Container Registry のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールには Amazon ECR リソースを作成または変更するアクセス許可がありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

Amazon ECR が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「Amazon Elastic Container Registry のアクション、リソース、および条件キー」を参照してください。

JSON ポリシードキュメントのこれらの例を使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーのベストプラクティス
- Amazon ECR コンソールの使用
- ユーザーが自分のアクセス許可を表示できるようにする
- 1 つの Amazon ECR リポジトリにアクセスする

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内で誰かが Amazon ECR リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションでは、 AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。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 または ジョブ機能のAWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

Amazon ECR コンソールの使用

Amazon Elastic Container Registry コンソールにアクセスするには、最小限の許可のセットが必要です。これらのアクセス許可により、 AWS アカウントの Amazon ECR リソースの詳細を一覧表示および表示できます。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

これらのエンティティが引き続き Amazon ECR コンソールを使用できるようにするには、エンティティに AmazonEC2ContainerRegistryReadOnly AWS 管理ポリシーを追加します。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonElasticContainerRegistryPublicReadOnly」を参照してください。

AWS CLI または AWS API のみを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。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーが自分のアクセス許可を表示できるようにする

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

1 つの Amazon ECR リポジトリにアクセスする

この例では、 AWS アカウントのユーザーに Amazon ECR リポジトリの 1 つである へのアクセス権を付与しますmy-repo。また、ユーザーがイメージをプッシュ、プル、リスト表示できるようにします。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"GetAuthorizationToken",
         "Effect":"Allow",
         "Action":[
            "ecr:GetAuthorizationToken"
         ],
         "Resource":"*"
      },
      {
         "Sid":"ManageRepositoryContents",
```

```
        "Effect":"Allow",
        "Action":[
               "ecr:BatchCheckLayerAvailability",
               "ecr:GetDownloadUrlForLayer",
               "ecr:GetRepositoryPolicy",
               "ecr:DescribeRepositories",
               "ecr:ListImages",
               "ecr:DescribeImages",
               "ecr:BatchGetImage",
               "ecr:InitiateLayerUpload",
               "ecr:UploadLayerPart",
               "ecr:CompleteLayerUpload",
               "ecr:PutImage"
        ],
        "Resource":"arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
      }
   ]
}
```

タグベースのアクセスコントロールを使用する

Amazon ECR `CreateRepository` API アクションでは、リポジトリ作成時にタグを指定することができます。詳細については、「Amazon ECR でのプライベートリポジトリのタグ付け」を参照してください。

作成時にユーザーがリポジトリにタグ付けするには、そのリソースを作成するアクションを使用するためのアクセス許可が必要です (例: `ecr:CreateRepository`)。タグがリソース作成アクションで指定されている場合、Amazon は `ecr:CreateRepository` アクションで追加の認可を実行してユーザーがタグを作成するアクセス許可を持っているかどうかを確認します。

タグベースのアクセスコントロールは、IAM ポリシーを介して使用できます。以下は例です。

次のポリシーによって、ユーザーは、`key=environment,value=dev` としてリポジトリを作成またはタグ付けすることのみ許可されます。

JSON

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

```
        {
            "Sid": "AllowCreateTaggedRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": "dev"
                }
            }
        },
        {
            "Sid": "AllowTagRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": "dev"
                }
            }
        }
    ]
}
```

次のポリシーでは、key=environment,value=prod としてタグ付けされている場合を除き、ユーザーはすべてのリポジトリからイメージをプルできます。

JSON

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

```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecr:ResourceTag/environment": "prod"
                }
            }
        }
    ]
}
```

AWS Amazon Elastic Container Registry の マネージドポリシー

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、多くの一般的なユースケースにアクセス許可を提供するように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS マネージドポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。 AWS は、新しい が起動されるか、新しい API オペレーション AWS のサービス が既存のサービスで使用できるようになったときに、 AWS マネージドポリシーを更新する可能性が高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

Amazon ECR は、IAM ID または Amazon EC2 インスタンスにアタッチできる複数の管理ポリシーを提供しています。これらの管理ポリシーを使用すると、Amazon ECR リソースおよび API オペレーションへのアクセスをさまざまな制御レベルで許可できます。これらのポリシーに記載されてい

る各 API オペレーションの詳細については、Amazon Elastic Container Registry API リファレンスの「アクション」を参照してください。

トピック

- AmazonEC2ContainerRegistryFullAccess
- AmazonEC2ContainerRegistryPowerUser
- Amazon EC2コンテナレジストリプルオンリー
- AmazonEC2ContainerRegistryReadOnly
- AWSECRPullThroughCache_ServiceRolePolicy
- ECRReplicationServiceRolePolicy
- ECRTemplateServiceRolePolicy
- AWS マネージドポリシーに対する Amazon ECR の更新

AmazonEC2ContainerRegistryFullAccess

`AmazonEC2ContainerRegistryFullAccess` ポリシーを IAM アイデンティティにアタッチできます。このポリシーは、Amazon ECR リソースへの管理アクセス権を付与し、Amazon ECR が統合されている AWS サービスへの IAM ID (ユーザー、グループ、ロールなど) アクセス権を付与して、すべての Amazon ECR 機能を使用します。このポリシーを使用すると、 AWS マネジメントコンソールで利用可能な Amazon ECR のすべての機能にアクセスできます。

このポリシーの許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonEC2ContainerRegistryFullAccess」を参照してください。

AmazonEC2ContainerRegistryPowerUser

`AmazonEC2ContainerRegistryPowerUser` ポリシーを IAM アイデンティティにアタッチできます。このポリシーは、IAM ユーザーがリポジトリに対する読み取りと書き込みを行う管理権限を付与しますが、IAM ユーザーは、リポジトリを削除することや、適用されるポリシードキュメントを変更することはできません。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonEC2ContainerRegistryPowerUser」を参照してください。

Amazon EC2コンテナレジストリプルオンリー

`AmazonEC2ContainerRegistryPullOnly` ポリシーを IAM アイデンティティにアタッチできます。このポリシーは、Amazon ECR からコンテナイメージをプルするためのアクセス許可を付与し

ます。レジストリでプルスルーキャッシュが有効になっている場合、アップストリームレジストリからイメージをインポートするためのプルも許可されます。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonEC2ContainerRegistryPullOnly」を参照してください。

AmazonEC2ContainerRegistryReadOnly

AmazonEC2ContainerRegistryReadOnly ポリシーを IAM アイデンティティにアタッチできます。このポリシーでは、Amazon ECR に対する読み取り専用アクセスを付与します。リポジトリおよびリポジトリ内のイメージをリストすることができます。また、Docker CLI を使用して Amazon ECR からイメージをプルすることもできます。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonEC2ContainerRegistryReadOnly」を参照してください。

AWSECRPullThroughCache_ServiceRolePolicy

AWSECRPullThroughCache_ServiceRolePolicy マネージド IAM ポリシーを IAM エンティティにアタッチすることはできません。このポリシーは、プルスルーキャッシュワークフローを通じて Amazon ECR がイメージをリポジトリにプッシュすることを許可する、サービスにリンクされたロールにアタッチされます。詳細については、「プルスルーキャッシュの Amazon ECR サービスにリンクされたロール」を参照してください。

ECRReplicationServiceRolePolicy

ECRReplicationServiceRolePolicy マネージド IAM ポリシーを IAM エンティティにアタッチすることはできません。このポリシーは、サービスにリンクされたロールにアタッチされ、ユーザーに代わって Amazon ECR がアクションを実行することを許可します。詳細については、「Amazon ECR でのサービスにリンクされたロールの使用」を参照してください。

ECRTemplateServiceRolePolicy

ECRTemplateServiceRolePolicy マネージド IAM ポリシーを IAM エンティティにアタッチすることはできません。このポリシーは、サービスにリンクされたロールにアタッチされ、ユーザーに代わって Amazon ECR がアクションを実行することを許可します。詳細については、「Amazon ECR でのサービスにリンクされたロールの使用」を参照してください。

AWS マネージドポリシーに対する Amazon ECR の更新

このサービスがこれらの変更の追跡を開始した時点からの Amazon ECR 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有効にするには、Amazon ECR ドキュメントの履歴ページから RSS フィードをサブスクライブしてください。

変更	説明	日付
プルスルーキャッシュの Amazon ECR サービスにリンクされたロール - 既存ポリシーへの更新	Amazon ECR は、新しいアクセス許可を `AWSECRPullThroughCache_ServiceRolePolicy` ポリシーに追加しました。これらのアクセス許可では、Amazon ECR が ECR プライベートレジストリからイメージをプルすることが許可されます。これは、プルスルーキャッシュルールを使用して別の Amazon ECR プライベートレジストリからイメージをキャッシュするときに必要です。	2025 年 3 月 12 日
AmazonEC2ContainerRegistryPullOnly – 新しいポリシー	Amazon ECR に、プル専用アクセス許可を Amazon ECR に付与する新しいポリシーが追加されました。	2024 年 10 月 10 日
ECRTemplateServiceRolePolicy – 新しいポリシー	Amazon ECR は新しいポリシーを追加しました。このポリシーは、リポジトリ作成テンプレート機能の `ECRTemplateServiceRolePolicy` サービスリンクロールに関連付けられます。	2024 年 6 月 20 日

変更	説明	日付
AWSECRPullThroughCache_ServiceRolePolicy — 既存のポリシーへの更新	Amazon ECR は、新しいアクセス許可を AWSECRPullThroughCache_ServiceRolePolicy ポリシーに追加しました。これらの権限により、Amazon ECR は Secrets Manager シークレットの暗号化されたコンテンツを取得できます。これは、プルスルーキャッシュルールを使用して、認証が必要なアップストリームレジストリからイメージをキャッシュする場合に必要です。	2023 年 11 月 15 日
AWSECRPullThroughCache_ServiceRolePolicy — 新しいポリシー	Amazon ECR は新しいポリシーを追加しました。このポリシーは、プルスルーキャッシュ機能の AWSServiceRoleForECRPullThroughCache サービスにリンクされたロールに関連付けられます。	2021 年 11 月 29 日
ECRReplicationServiceRolePolicy — 新しいポリシー	Amazon ECR は新しいポリシーを追加しました。このポリシーは、レプリケーション機能の AWSServiceRoleForECRReplication サービスにリンクされたロールに関連付けられます。	2020 年 12 月 4 日

変更	説明	日付
AmazonEC2ContainerRegistryFullAccess — 既存のポリシーに対する更新	Amazon ECR は、新しいアクセス許可を AmazonEC2ContainerRegistryFullAccess ポリシーに追加しました。これらのアクセス許可により、プリンシパルは Amazon ECR サービスにリンクされたロールを作成できるようになります。	2020 年 12 月 4 日
AmazonEC2ContainerRegistryReadOnly — 既存のポリシーに対する更新	Amazon ECR は、ライフサイクルポリシーの読み取り、タグの一覧表示、イメージのスキャン結果の記述をプリンシパルに許可する新しいアクセス許可を AmazonEC2ContainerRegistryReadOnly ポリシーに追加しました。	2019 年 12 月 10 日
AmazonEC2ContainerRegistryPowerUser — 既存のポリシーに対する更新	Amazon ECR は、新しいアクセス許可を AmazonEC2ContainerRegistryPowerUser ポリシーに追加しました。これにより、プリンシパルは、ライフサイクルポリシーの読み取り、タグの一覧表示、イメージのスキャン結果の記述を行うことができるようになります。	2019 年 12 月 10 日

変更	説明	日付
AmazonEC2ContainerRegistryFullAccess — 既存のポリシーに対する更新	Amazon ECR は、新しいアクセス許可を `AmazonEC2ContainerRegistryFullAccess` ポリシーに追加しました。これにより、プリンシパルは CloudTrail によってキャプチャされた管理イベントまたは AWS CloudTrail Insights イベントを検索できます。	2017 年 11 月 10 日
AmazonEC2ContainerRegistryReadOnly — 既存のポリシーに対する更新	Amazon ECR は、新しいアクセス許可を `AmazonEC2ContainerRegistryReadOnly` ポリシーに追加しました。これにより、プリンシパルは Amazon ECR イメージを記述できるようになります。	2016 年 10 月 11 日
AmazonEC2ContainerRegistryPowerUser — 既存のポリシーに対する更新	Amazon ECR は、新しいアクセス許可を `AmazonEC2ContainerRegistryPowerUser` ポリシーに追加しました。これにより、プリンシパルは Amazon ECR イメージを記述できるようになります。	2016 年 10 月 11 日

変更	説明	日付
AmazonEC2ContainerRegistryReadOnly – 新しいポリシー	Amazon ECR は、Amazon ECR に読み取り専用アクセス権限を付与する新しいポリシーを追加しました。これらのアクセス許可には、リポジトリおよびリポジトリ内のイメージをリストすることが含まれます。また、Docker CLI を使用して Amazon ECR からイメージをプルすることもできます。	2015 年 12 月 21 日
AmazonEC2ContainerRegistryPowerUser — 新しいポリシー	Amazon ECR は、ユーザーがリポジトリに対する読み取りと書き込みを行う管理権限を付与する新しいポリシーを追加しました。しかし、ユーザーは、リポジトリを削除することや、適用されるポリシードキュメントを変更することはできません。	2015 年 12 月 21 日
AmazonEC2ContainerRegistryFullAccess – 新しいポリシー	Amazon ECR は新しいポリシーを追加しました。このポリシーは、Amazon ECR へのフルアクセスを付与します。	2015 年 12 月 21 日
Amazon ECR が変更の追跡を開始しました。	Amazon ECR 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 24 日

Amazon ECR でのサービスにリンクされたロールの使用

Amazon Elastic Container Registry (Amazon ECR) 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て、レプリケーションとプルスルーキャッシュ機能

を使用するために必要なアクセス許可を提供します。サービスにリンクされたロールは、Amazon ECR に直接リンクされた特殊な IAM ロールです。サービスにリンクされたロールは、Amazon ECR で事前定義されています。これには、サービスがプライベートレジストリのレプリケーションとプルスルーキャッシュ機能をサポートするために必要なすべてのアクセス許可が含まれます。レジストリのレプリケーションまたはプルスルーキャッシュを構成すると、サービスにリンクされたロールが自動的に作成されます。詳細については、「Amazon ECR でのプライベートレジストリ設定」を参照してください。

サービスにリンクされたロールを使用すると、Amazon ECR を使用したレプリケーションとプルスルーキャッシュの設定が簡単になります。この理由は、このロールを使用することにより、必要なアクセス許可をすべて手動で追加する必要がなくなるためです。サービスにリンクされたロールのアクセス許可は、Amazon ECR により定義されます。特に指定されている場合を除き、そのロールを引き受けることができるのは Amazon ECR のみです。定義された許可には、信頼ポリシーと許可ポリシーが含まれます。アクセス許可ポリシーを他の IAM エンティティにアタッチすることはできません。

対応するサービスにリンクされたロールを削除できるのは、先にレジストリでレプリケーションまたはプルスルーキャッシュのいずれかを無効にした後のみです。これにより、これらの機能に対して Amazon ECR で必要なアクセス許可を誤って削除することがなくなります。

サービスにリンクされたロールをサポートす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。このリンク先のページで、「サービスにリンクされたロール」列が「あり」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、[あり] リンクを選択します。

トピック

- Amazon ECR のサービスにリンクされたロールがサポートされるリージョン
- レプリケーション用の Amazon ECR サービスにリンクされたロール
- プルスルーキャッシュの Amazon ECR サービスにリンクされたロール
- リポジトリ作成テンプレート用の Amazon ECR サービスリンクロール

Amazon ECR のサービスにリンクされたロールがサポートされるリージョン

Amazon ECR では、このサービスを利用できるすべてのリージョンで、サービスにリンクされたロールの使用がサポートされています。Amazon ECR リージョンの可用性については、「AWS リージョンとエンドポイント」を参照してください。

レプリケーション用の Amazon ECR サービスにリンクされたロール

Amazon ECR は、AWSServiceRoleForECRReplication という名前のサービスリンクロールを使用します。これは、Amazon ECR が複数のアカウント間でイメージをレプリケートすることを許可します。

Amazon ECR でのサービスにリンクされたロールのアクセス許可

サービスにリンクされた AWSServiceRoleForECRReplication ロールは、以下のサービスを信頼してロールを引き受けます。

- `replication.ecr.amazonaws.com`

以下の `ECRReplicationServiceRolePolicy` ロールのアクセス許可ポリシーは、リソースに対して以下のアクションを使用することを Amazon ECR に許可します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> `ReplicateImage` は、Amazon ECR がレプリケーションに使用する内部 API で、直接呼び出すことはできません。

アクセス許可を設定して、IAM エンティティ (ユーザー、グループ、ロールなど) がサービスリンクロールの作成、編集、削除を行うことを許可する必要があります。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アクセス許可」を参照してください。

Amazon ECR でのサービスにリンクされたロールの作成

Amazon ECR 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 でレジストリのレプリケーション設定を構成すると、Amazon ECR 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。

このサービスにリンクされたロールを削除した後に、そのロールを再作成する必要がある場合は、同じプロセスを使用してアカウントでロールを再作成することができます。レジストリのレプリケーション設定を構成すると、サービスにリンクされたロールAmazon ECR が Amazon ECR によって再度作成されます。

Amazon ECR でのサービスにリンクされたロールの編集

Amazon ECR では、サービスにリンクされた AWSServiceRoleForECRReplication ロールを手動で削除することはできません。サービスリンクロールを作成すると、多くのエンティティによってロールが参照され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 を使用したロール記述の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon ECR で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やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、アクティブにモニタリングまたはメンテナンスされない未使用の エンティティがなくなります。ただし、サービスにリンクされたロールを手動で削除するには、すべてのリージョンでレジストリのレプリケーション設定を削除する必要があります。

> **Note**
>
> Amazon ECR サービスがロールを使用しているときにリソースを削除しようとすると、削除アクションが失敗することがあります。その場合は、数分待ってから再試行してください。

サービスにリンクされた AWSServiceRoleForECRReplication ロールが使用している Amazon ECR リソースを削除するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。

2. ナビゲーションバーから、レプリケーション設定が有効なリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ) を選択します。
4. [Private registry] (プライベートレジストリ) ページの [Replication configuration] (レプリケーション設定) セクションで、[Edit] (編集) をクリックします。
5. すべてのレプリケーションルールを削除するには、[Delete all] (すべて削除) を選択します。このステップには確認が必要です。

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、AWSServiceRoleForECRReplication 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、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

プルスルーキャッシュの Amazon ECR サービスにリンクされたロール

Amazon ECRは、Amazon ECR がユーザーに代わってプルスルーキャッシュワークフローを使用してリポジトリにイメージをプッシュするためのアクセス許可を付与する、AWSServiceRoleForECRPullThroughCache という名前のサービスにリンクされたロールを使用します。プルスルーの詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。

Amazon ECR でのサービスにリンクされたロールのアクセス許可

AWSServiceRoleForECRPullThroughCache サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `pullthroughcache.ecr.amazonaws.com`

アクセス許可の詳細

`AWSECRPullThroughCache_ServiceRolePolicy` 許可ポリシーは、サービスにリンクされたロールにアタッチされます。この管理ポリシーは Amazon ECR に以下のアクションを実行する許可を付与します。詳細については、「`AWSECRPullThroughCache_ServiceRolePolicy`」を参照してください。

- `ecr` — Amazon ECR サービスがプライベートリポジトリにイメージをプルおよびプッシュすることを許可します。

- secretsmanager:GetSecretValue – Amazon ECR サービスが AWS Secrets Manager シークレットの暗号化されたコンテンツを取得できるようにします。これは、プルスルーキャッシュルールを使用して、プライベートレジストリの認証が必要なアップストリームレジストリからイメージをキャッシュする場合に必要です。この許可は、ecr-pullthroughcache/ という名前のプレフィックスが付いたシークレットのみに適用されます。

AWSECRPullThroughCache_ServiceRolePolicy ポリシーには、次の JSON が含まれます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ECR",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage",
                "ecr:BatchGetImage",
                "ecr:BatchImportUpstreamImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetImageCopyStatus"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManager",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ecr-pullthroughcache/
*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
```

```
                }
            }
        }
    ]
}
```

アクセス許可を設定して、IAM エンティティ (ユーザー、グループ、ロールなど) がサービスリンクロールの作成、編集、削除を行うことを許可する必要があります。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役割のアクセス許可」を参照してください。

Amazon ECR でのサービスにリンクされたロールの作成

プルスルーキャッシュの Amazon ECR 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 でプライベートレジストリのプルスルーキャッシュルールを作成すると、Amazon ECR 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。

このサービスにリンクされたロールを削除した後に、そのロールを再作成する必要がある場合は、同じプロセスを使用してアカウントでロールを再作成することができます。プライベートレジストリーのプルスルーキャッシュルールを作成すると、サービスにリンクされたロールがまだ存在しない場合は、Amazon ECR によって自動的に作成されます。

Amazon ECR でのサービスにリンクされたロールの編集

Amazon ECR は、AWSServiceRoleForECRPullThroughCache サービスにリンクされたロールの手動での編集を許可しません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照され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 を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon ECR で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やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、アクティブにモニタリングまたはメンテナンスされない未使用のエンティティがなくなります。ただし、サービスにリンクされたロールを手動で削除するには、すべてのリージョンでレジストリのプルスルーキャッシュルールを削除する必要があります。

> **Note**
>
> Amazon ECR サービスがロールをまだ使用しているときにリソースを削除しようとすると、削除アクションが失敗することがあります。その場合は、数分待ってから再試行してください。

AWSServiceRoleForECRPullThroughCache サービスリンクロールが使用している Amazon ECR リソースを削除するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、プルスルーキャッシュルールを作成するリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ) を選択します。
4. [Private registry] (プライベートレジストリ) ページの [Pull through cache configuration] (プルスルーキャッシュの設定) セクションで、[Edit] (編集) をクリックします。
5. 作成したプルスルーキャッシュルールごとに、ルールを選択し、[Delete rule] (ルールを削除) を選択します。

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、AWSServiceRoleForECRPullThroughCache 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、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

リポジトリ作成テンプレート用の Amazon ECR サービスリンクロール

Amazon ECR は、AWSServiceRoleForECRTemplate という名前のサービスリンクロールを使用します。これは、リポジトリ作成テンプレートのアクションを完了するために Amazon ECR がユーザーに代わってアクションを実行するためのアクセス許可を付与します。

Amazon ECR でのサービスにリンクされたロールのアクセス許可

AWSServiceRoleForECRTemplate サービスリンクロールは、次のサービスを信頼してロールを引き受けます。

- `ecr.amazonaws.com`

アクセス許可の詳細

ECRTemplateServiceRolePolicy 許可ポリシーは、サービスにリンクされたロールにアタッチされます。この管理ポリシーは、ユーザーに代わってリポジトリ作成アクションを実行するためのアクセス許可を Amazon ECR に付与します。

ECRTemplateServiceRolePolicy ポリシーには、次の JSON が含まれます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
     "Sid": "CreateRepositoryWithTemplate",
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository"
            ],
            "Resource": "*"
        }
    ]
}
```

アクセス許可を設定して、IAM エンティティ (ユーザー、グループ、ロールなど) がサービスリンクロールの作成、編集、削除を行うことを許可する必要があります。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役割のアクセス許可」を参照してください。

Amazon ECR でのサービスにリンクされたロールの作成

リポジトリ作成テンプレート用に Amazon ECR サービスリンク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 でプライベートレジストリのリポジトリ作成テンプレートルールを作成すると、Amazon ECR 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。

このサービスにリンクされたロールを削除した後に、そのロールを再作成する必要がある場合は、同じプロセスを使用してアカウントでロールを再作成することができます。プライベートレジストリ用のリポジトリ作成ルールを作成すると、Amazon ECR によってサービスリンクロールが自動的に作成されます (まだ存在しない場合)。

Amazon ECR でのサービスにリンクされたロールの編集

Amazon ECR では、AWSServiceRoleForECRTemplate サービスリンクロールを手動で編集することは許可されません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照され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 を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon ECR で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やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、アクティブにモニタリングまたはメンテナンスされない未使用のエンティティがなくなります。ただし、サービスリンクロールを手動で削除するには、すべてのリージョンでレジストリ用のリポジトリ作成ルールを削除する必要があります。

> **Note**
>
> Amazon ECR サービスがロールをまだ使用しているときにリソースを削除しようとすると、削除アクションが失敗することがあります。その場合は、数分待ってから再試行してください。

AWSServiceRoleForECRTemplate サービスリンクロールが使用している Amazon ECR リソースを削除する場合

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、リポジトリ作成ルールが作成されているリージョンを選択します。
3. ナビゲーションペインで、[Private registry] (プライベートレジストリ) を選択します。
4. [プライベートレジストリ] ページの [リポジトリ作成テンプレート] セクションで、[編集] を選択します。
5. 作成したリポジトリ作成ルールごとに、ルールを選択し、[ルールの削除] を選択します。

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、AWSServiceRoleForECRTemplate 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、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

Amazon Elastic Container Registry の アイデンティティとアクセスのトラブルシューティング

次の情報は、Amazon ECR と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- Amazon ECR でアクションを実行する権限がない
- iam:PassRole を実行する権限がない
- 自分の 以外のユーザーに Amazon ECR リソース AWS アカウント へのアクセスを許可したい

Amazon ECR でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な ecr:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 ecr:GetWidget on resource: my-example-widget
```

この場合、ecr:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がない

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon ECR にロールを渡せるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon ECR でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに Amazon ECR リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon ECR がこれらの機能をサポートしているかどうかについては、「Amazon Elastic Container Registry と IAM が連動するしくみ」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、IAM ユーザーガイド の IAM でのクロスアカウントのリソースへのアクセス を参照してください。

Amazon ECR でのデータ保護

AWS 責任共有モデル、Amazon Elastic Container Service 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 Amazon ECS AWS CLIまたは他の AWS のサービス を操作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。

トピック

- 保管中の暗号化

保管中の暗号化

> **Important**
>
> AWS KMS (DSSE-KMS) を使用した二層式サーバー側の暗号化は、 AWS GovCloud (US) リージョンでのみ使用できます。

Amazon ECR は、Amazon ECR が管理する Amazon S3 バケットにイメージを保存します。デフォルトでは、Amazon ECR は、Amazon S3 が管理する暗号化キーによるサーバー側暗号化を使用し、保管中のデータが AES-256 暗号化アルゴリズムで暗号化されます。これは、お客様によるアクションを必要とせず、追加料金なしで提供されます。詳細については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 が管理する暗号化キーによるサーバー側の暗号化 (SSE-S3) を使用したデータの保護」を参照してください。

Amazon ECR リポジトリの暗号化をより詳細に制御するには、 AWS Key Management Service () に保存されている KMS キーでサーバー側の暗号化を使用できますAWS KMS。を使用して AWS KMS データを暗号化する場合、Amazon ECR によって管理 AWS マネージドキーされるデフォルトを使用するか、独自の KMS キー (カスタマーマネージドキーと呼ばれます) を指定できます。詳細については、Amazon Simple Storage Service ユーザーガイドのAWS KMS 「 (SSE-KMS) に保存されている KMS キーによる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。

AWS KMS () による二層式サーバー側の暗号化を使用して、Amazon ECR イメージに 2 つの暗号化レイヤーを適用できますDSSE-KMS。 DSSE-KMSオプションは に似ていますがSSE-KMS、1 つのレイヤーではなく 2 つの個別の暗号化レイヤーを適用します。詳細については、AWS KMS 「キーによる二層式サーバー側の暗号化の使用 (DSSE-KMS)」を参照してください。

各 Amazon ECR リポジトリには、リポジトリの作成時に設定される暗号化設定があります。リポジトリごとに異なる暗号化設定を使用できます。詳細については、「イメージを保存するための Amazon ECR プライベートリポジトリの作成」を参照してください。

AWS KMS 暗号化を有効にしてリポジトリを作成すると、KMS キーを使用してリポジトリの内容を暗号化します。さらに、Amazon ECR は、Amazon ECR リポジトリを AWS KMS 被付与者プリンシパルとして KMS キーに権限を追加します。

Amazon ECR が AWS KMS と統合してレポジトリを暗号化および復号化する方法の高レベルの概要を以下に示します。

1. リポジトリを作成すると、Amazon ECR は DescribeKey 呼び出しを に送信 AWS KMS し、暗号化設定で指定された KMS キーの Amazon リソースネーム (ARN) を検証して取得します。
2. Amazon ECR は、2 つの CreateGrant リクエストを AWS KMS に送信して KMS キーに許可を作成し、Amazon ECR がデータキーを使用してデータを暗号化および復号できるようにします。
3. イメージをプッシュすると、イメージレイヤーとマニフェストの暗号化に使用する KMS キー AWS KMS を指定する GenerateDataKey リクエストが に対して行われます。
4. AWS KMS は新しいデータキーを生成し、指定された KMS キーで暗号化し、暗号化されたデータキーを送信してイメージレイヤーメタデータとイメージマニフェストとともに保存します。
5. イメージをプルすると、暗号化されたデータキーを指定して AWS KMS、復号リクエストが に対して行われます。
6. AWS KMS は暗号化されたデータキーを復号し、復号されたデータキーを Amazon S3 に送信します。
7. データキーは、イメージレイヤーをプルする前にイメージレイヤーを復号化するために使用されます。
8. リポジトリが削除されると、Amazon ECR は 2 つの RetireGrant リクエストを に送信 AWS KMS して、リポジトリ用に作成された許可を廃止します。

考慮事項

Amazon ECR で AWS KMS ベースの暗号化 (SSE-KMS または DSSE-KMS) を使用する場合は、次の点を考慮する必要があります。

- KMS 暗号化を使用して Amazon ECR リポジトリを作成し、KMS キーを指定しない場合、Amazon ECR は`aws/ecr`デフォルトでエイリアス AWS マネージドキー で を使用します。この KMS キーは、KMS 暗号化を有効にしたリポジトリを初めて作成するときに、アカウントに作成されます。
- リポジトリの作成後にリポジトリの暗号化設定を変更することはできません。
- 独自の KMS キーで KMS 暗号化を使用する場合、そのキーはリポジトリと同じリージョンに存在する必要があります。

- お客様の代わりに Amazon ECR が作成する許可は取り消さないでください。アカウントの AWS KMS キーを使用するアクセス許可を Amazon ECR に付与する許可を取り消すと、Amazon ECR はこのデータにアクセスしたり、リポジトリにプッシュされた新しいイメージを暗号化したり、プル時に復号したりすることはできません。Amazon ECR の許可を取り消すと、変更がすぐに適用されます。アクセス権限を取り消すには、許可を取り消すのではなく、リポジトリを削除します。リポジトリを削除すると、Amazon ECR がユーザーに代わって許可を取り消します。
- AWS KMS キーの使用にはコストがかかります。詳細については、AWS Key Management Service 料金表を参照してください。
- 二層式サーバー側の暗号化の使用にはコストがかかります。詳細については、Amazon ECR の料金を参照してください。

必要な IAM 許可

AWS KMSを使用するサーバー側の暗号化が有効な Amazon ECR リポジトリを作成または削除する場合、必要なアクセス許可は、使用する特定の KMS キーに応じて異なります。

Amazon ECR AWS マネージドキー の を使用する場合に必要な IAM アクセス許可

デフォルトでは、Amazon ECR リポジトリで AWS KMS 暗号化が有効になっていても KMS キーが指定されていない場合、Amazon ECR AWS マネージドキー の が使用されます。Amazon ECR の AWSマネージド KMS キーを使用してリポジトリを暗号化する場合、リポジトリを作成するアクセス許可を持つプリンシパルは、リポジトリで AWS KMS 暗号化を有効にすることもできます。ただし、リポジトリを削除する IAM プリンシパルには、`kms:RetireGrant` アクセス許可が必要です。これにより、リポジトリの作成時に AWS KMS キーに追加された許可が廃止されます。

次の IAM ポリシーの例をインラインポリシーとしてユーザーに追加すると、暗号化が有効なリポジトリを削除するのに必要な最小限のアクセス許可がユーザーに付与されます。リポジトリの暗号化に使用する KMS キーは、リソースパラメータを使用して指定できます。

JSON

```
{
    "Version":"2012-10-17",
    "Id": "ecr-kms-permissions",
    "Statement": [
        {
            "Sid": "AllowAccessToRetireTheGrantsAssociatedWithTheKey",
            "Effect": "Allow",
```

```
            "Action": [
                "kms:RetireGrant"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/b8d9ae76-080c-4043-92EXAMPLE"
        }
    ]
}
```

カスタマーマネージドキーを使用する場合に必要な IAM アクセス許可

カスタマーマネージドキーを使用して AWS KMS 暗号化を有効にしたリポジトリを作成する場合、リポジトリを作成するユーザーまたはロールの KMS キーポリシーと IAM ポリシーの両方に必要なアクセス許可があります。

独自の KMS キーを作成する場合、 AWS KMS が作成するデフォルトのキーポリシーを使用するか、独自のキーポリシーを指定することができます。アカウント所有者がカスタマーマネージドキーを管理できるように、KMS キーのキーポリシーでは、アカウントのルートユーザーのすべての AWS KMS アクションを許可する必要があります。追加のスコープ付きアクセス許可をキーポリシーに追加することもできますが、少なくともルートユーザーには KMS キーを管理するためのアクセス許可が必要です。Amazon ECR から実行されるリクエストにのみ KMS を使用するには、kms:ViaService 条件キーを `ecr.<region>.amazonaws.com` 値で使用できます。

次のキーポリシーの例では、KMS キーを所有する AWS アカウント (ルートユーザー) に KMS キーへのフルアクセスを許可します。このキーポリシーの例の詳細については、「 AWS Key Management Service デベロッパーガイド」の AWS 「アカウントへのアクセスを許可」および「IAM ポリシーを有効にする」を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Id": "ecr-key-policy",
    "Statement": [
        {
            "Sid": "EnableIAMUserPermissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
```

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

リポジトリを作成する IAM ユーザー、IAM ロール、または AWS アカウントには`kms:CreateGrant`、必要な Amazon ECR アクセス`kms:DescribeKey`許可に加えて、、`kms:RetireGrant`、および アクセス許可が必要です。

> **Note**
>
> リポジトリを作成するユーザーまたはロールの IAM ポリシーに `kms:RetireGrant` アクセス許可を追加する必要があります。`kms:CreateGrant` と `kms:DescribeKey` のアクセス許可は、KMS キーのキーポリシー、またはリポジトリを作成するユーザーまたはロールの IAM ポリシーに追加できます。アクセス AWS KMS 許可の仕組みの詳細については、「AWS Key Management Service デベロッパーガイド」のAWS KMS 「API アクセス許可: アクションとリソースのリファレンス」を参照してください。

次の IAM ポリシーの例をインラインポリシーとしてユーザーに追加すると、暗号化が有効なリポジトリの作成、および作業が終了したときにリポジトリの削除を行うのに必要な最小限のアクセス許可がユーザーに付与されます。リポジトリの暗号化に使用する AWS KMS key は、リソースパラメータを使用して指定できます。

JSON

```
{
    "Version":"2012-10-17",
    "Id": "ecr-kms-permissions",
    "Statement": [
        {
            "Sid":
 "AllowAccessToCreateAndRetireTheGrantsAssociatedWithTheKeyAsWellAsDescribeTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:RetireGrant",
                "kms:DescribeKey"
```

```
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/b8d9ae76-080c-4043-92EXAMPLE"
        }
    ]
}
```

リポジトリの作成時にコンソールに KMS キーを一覧表示できるようにする

Amazon ECR コンソールを使用してリポジトリを作成すると、リポジトリの暗号化を有効にするときに、ユーザーがリージョンでカスタマーマネージド KMS キーを一覧表示するアクセス許可を付与できます。次の IAM ポリシーの例は、コンソールを使用するときに KMS キーとエイリアスを一覧表示するために必要なアクセス許可を示しています。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:ListKeys",
      "kms:ListAliases",
      "kms:DescribeKey"
    ],
    "Resource": "*"
  }
}
```

との Amazon ECR インタラクションのモニタリング AWS KMS

を使用して AWS CloudTrail 、Amazon ECR が AWS KMS ユーザーに代わって に送信するリクエストを追跡できます。CloudTrail ログのログエントリには、より簡単に識別できるように暗号化コンテキストキーが含まれています。

Amazon ECR 暗号化コンテキスト

暗号化コンテキスト は、任意非シークレットデータを含むキーと値のペアのセットです。データを暗号化するリクエストに暗号化コンテキストを含めると、 は暗号化コンテキストを暗号化された

データに AWS KMS 暗号化バインドします。データを復号するには、同じ暗号化コンテキストに渡す必要があります。

GenerateDataKey および Decrypt リクエストでは AWS KMS、Amazon ECR は、使用するリポジトリと Amazon S3 バケットを識別する 2 つの名前と値のペアを持つ暗号化コンテキストを使用します。以下の例ではこれを示しています。名前は変わりませんが、組み合わされた暗号化コンテキストの値は、値ごとに異なります。

```
"encryptionContext": {
    "aws:s3:arn": "arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-ba987BUCKET1/
sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df",
    "aws:ecr:arn": "arn:aws:ecr:us-west-2:111122223333:repository/repository-name"
}
```

暗号化コンテキストを使用して、AWS CloudTrail や Amazon CloudWatch Logs などの監査レコードやログで、ポリシーや許可の認可の条件として、これらの暗号化オペレーションを識別できます。

Amazon ECR 暗号化コンテキストは、2 つの名前と値のペアで構成されます。

- aws:s3:arn – バケットを識別する最初の名前と値のペア。キーは、`aws:s3:arn` です。値は、Amazon S3 バケットの Amazon リソースネーム (ARN)です。

  ```
  "aws:s3:arn": "ARN of an Amazon S3 bucket"
  ```

  たとえば、バケットの ARN が `arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-ba987BUCKET1/sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df` の場合、暗号化コンテキストには次のペアが含まれます。

  ```
  "arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-ba987BUCKET1/sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df"
  ```

- aws:ecr:arn – リポジトリの Amazon リソースネーム (ARN) を識別する 2 番目の名前と値のペア。キーは、`aws:ecr:arn` です。値は、リポジトリの ARNです。

  ```
  "aws:ecr:arn": "ARN of an Amazon ECR repository"
  ```

たとえば、リポジトリの ARN が arn:aws:ecr:*us-west-2*:*111122223333*:repository/*repository-name* の場合、暗号化コンテキストには次のペアが含まれます。

```
"aws:ecr:arn": "arn:aws:ecr:us-west-2:111122223333:repository/repository-name"
```

トラブルシューティング

コンソールで Amazon ECR リポジトリを削除するときに、リポジトリが正常に削除されても、Amazon ECR がリポジトリの KMS キーに追加された許可を使用停止できない場合、次のエラーが表示されます。

```
The repository [{repository-name}] has been deleted successfully but the grants created
 by the kmsKey [{kms_key}] failed to be retired
```

この場合、リポジトリの AWS KMS 許可を自分で廃止できます。

リポジトリの AWS KMS 許可を手動で廃止するには

1. リポジトリに使用される AWS KMS キーの許可を一覧表示します。key-id 値は、コンソールに表示されるエラーに含まれています。list-keys コマンドを使用して、アカウント内の特定のリージョンの AWS マネージドキー とカスタマーマネージド KMS キーの両方を一覧表示することもできます。

   ```
   aws kms list-grants \
        --key-id b8d9ae76-080c-4043-9237-c815bfc21dfc
        --region us-west-2
   ```

   出力には、リポジトリの Amazon リソースネーム (ARN) を含む EncryptionContextSubset が含まれます。これを使用して、使用停止にするのがキーに追加されたどの許可かを特定できます。GrantId 値は、次のステップで許可を使用停止するときに使用します。
2. リポジトリに追加された AWS KMS キーの各許可を廃止します。以前のステップの出力の許可の ID で *GrantId* の値を置き換えます。

   ```
   aws kms retire-grant \
        --key-id b8d9ae76-080c-4043-9237-c815bfc21dfc \
        --grant-id GrantId \
   ```

```
    --region us-west-2
```

Amazon Elastic Container Registry 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 プログラムによる対象範囲内」の「コンプライアンス」を参照し、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 Elastic Container Registry 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 Elastic Container Registry 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 がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して環境を 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 Amazon ECR にアクセスします。クライアントは次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、ほとんどの最新システムでサポートされています。

これらの API オペレーションは任意のネットワークの場所から呼び出すことができますが、Amazon ECR ではリソースベースのアクセスポリシーがサポートされているため、ソース IP アドレスに基づく制限を含めることができます。また、Amazon ECR ポリシーを使用して、特定の Amazon Virtual

Private Cloud (Amazon VPC) エンドポイントまたは特定の VPC からのアクセスを管理することもできます。これにより、実質的に、ネットワーク内の特定の VPC からのみ、特定の Amazon ECR リソースへの AWS ネットワークアクセスが分離されます。詳細については、「Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)

インターフェイス VPC エンドポイントを使用するように Amazon ECR を設定することで、VPC のセキュリティ体制を強化できます。VPC エンドポイントは AWS 、プライベート IP アドレス (IPv4 と IPv6 の両方) を介して Amazon ECR API にプライベートにアクセスできるテクノロジーである PrivateLink を利用しています。 APIs IPv4 IPv6 AWS PrivateLink は、VPC と Amazon ECR 間のすべてのネットワークトラフィックを Amazon ネットワークに制限します。インターネットゲートウェイ、NAT デバイス、または仮想プライベートゲートウェイは必要ありません。

AWS PrivateLink と VPC エンドポイントの詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイント」を参照してください。

Amazon ECR VPC エンドポイントに関する考慮事項

Amazon ECR の VPC エンドポイントを設定する前に、以下の考慮事項に注意してください。

- Amazon EC2 インスタンスでホストされる Amazon ECS タスクで Amazon ECR からプライベートイメージをプルできるようにするには、Amazon ECS 用のインターフェイス VPC エンドポイントを作成します。詳細については、Amazon Elastic Container Service デベロッパーガイドの「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。
- Amazon ECR からコンテナイメージをプルする Fargate でホストされる Amazon ECS タスクでは、条件キーをタスクのタスク実行 IAM ロールに追加することによって、タスクが使用する特定の VPC へのアクセス、およびサービスが使用する VPC エンドポイントへのアクセスを制限することができます。詳細については、Amazon Elastic Container Service デベロッパーガイドの「インターフェイスエンドポイントを介して Amazon ECR をプルする Fargate タスクの最適な IAM アクセス許可」を参照してください。
- VPC エンドポイントにアタッチされたセキュリティグループでは、VPC のプライベートサブネットから、ポート 443 で着信接続を許可する必要があります。
- Amazon ECR VPC エンドポイントは、デュアルスタック (IPv4 および IPv6) 接続をサポートしています。デュアルスタック VPC エンドポイントを作成すると、IPv4 プライベート IP アドレスと IPv6 プライベート IP アドレスの両方でトラフィックを処理できます。
- VPC エンドポイントは、米国東部 (バージニア北部) の AWS API SDK エンドポイントを介して Amazon ECR Public リポジトリをサポートします。

- VPC エンドポイントは、Amazon Route 53 を通じて AWS 提供される DNS のみをサポートします。独自の DNS を使用したい場合は、条件付き DNS 転送を使用できます。詳細については、Amazon VPC ユーザーガイド の DHCP Options Sets を参照してください。
- コンテナに Amazon S3 への既存の接続がある場合、Amazon S3 ゲートウェイエンドポイントを追加すると接続が一時的に中断される場合があります。この中断を回避するには、Amazon S3 ゲートウェイエンドポイントを使用する新しい VPC を作成してから、Amazon ECS クラスターとそのコンテナを新しい VPC に移行します。
- 初めてプルスルーキャッシュルールを使用してイメージをプルするとき、 AWS PrivateLink を使って、インターフェイス VPC エンドポイントを使用するように Amazon ECR を設定した場合、NAT ゲートウェイを使用して、同じ VPC 内にパブリックサブネットを作成し、プルが機能するように、プライベートサブネットから NAT ゲートウェイへのすべてのアウトバウンドトラフィックをインターネットにルーティングする必要があります。その後のイメージプルでは、これは必要ありません。詳細については、Amazon Virtual Private Cloud ユーザーガイドの「シナリオ: プライベートサブネットからインターネットにアクセスする」を参照してください。
- FIPS 140-3 検証済み暗号化モジュールを必要とするワークロードの場合、Amazon ECR は VPC エンドポイントの FIPS エンドポイントをサポートします。

Windows イメージに関する考慮事項

Windows オペレーティングシステムに基づくイメージには、ライセンスによって配布が制限されているアーティファクトが含まれます。デフォルトでは、Windows イメージを Amazon ECR リポジトリにプッシュすると、これらのアーティファクトを含むレイヤーは外部レイヤーと見なされるため、プッシュされません。アーティファクトが Microsoft によって提供されている場合、外部レイヤーは Microsoft Azure インフラストラクチャから取得されます。このため、コンテナがこれらの外部レイヤーを Azure からプルできるようにするには、VPC エンドポイントを作成する以外に、追加のステップが必要です。

Docker デーモンの `--allow-nondistributable-artifacts` フラグを使用して、Windowsイメージを Amazon ECR にプッシュするときに、この動作をオーバーライドすることができます。有効にすると、このフラグはライセンスされたレイヤーを Amazon ECR にプッシュします。これにより、Azure への追加アクセスを必要とすることなく、これらのイメージを VPC エンドポイント経由で Amazon ECR からプルすることができます。

> **Important**
>
> この `--allow-nondistributable-artifacts` フラグを使用しても、Windows コンテナベースイメージライセンスの条項に従う義務が排除されるわけではありません。したがっ

て、パブリックまたはサードパーティーによる再配布のために Windows コンテンツを投稿することはできません。お客様自身の環境内での使用は許可されています。

Docker インストールでこのフラグを使用できるようにするには、Docker デーモン設定ファイルを変更する必要があります。通常は Docker インストールに応じて、[Docker エンジン] セクションの設定または環境設定メニューで設定するか、直接 `C:\ProgramData\docker\config\daemon.json` ファイルを編集できます。

以下に示しているのは、必要な設定の例です。イメージをプッシュするリポジトリ URI に値を置き換えます。

```
{
    "allow-nondistributable-artifacts": [
        "111122223333.dkr.ecr.us-west-2.amazonaws.com"
    ]
}
```

Docker デーモン設定ファイルを変更したら、イメージをプッシュする前に Docker デーモンを再起動する必要があります。ベースレイヤーがリポジトリにプッシュされたことを確認して、プッシュが成功したことを確認します。

> **Note**
>
> Windows イメージのベースレイヤーは大きくなります。レイヤーサイズにより、プッシュ時間が長くなり、Amazon ECR でのこれらのイメージのストレージコストが増大します。これらの理由から、このオプションは、構築時間と継続的なストレージコストを削減することが厳密に要求される場合にのみ使用することをお勧めします。例えば、`mcr.microsoft.com/windows/servercore` イメージを Amazon ECR で圧縮すると、約 1.7 GiB のサイズになります。

Amazon ECR の VPC エンドポイントを作成する

Amazon ECR サービスの VPC エンドポイントを作成するには、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作成の手順を使用します。

Amazon ECR VPC エンドポイントは、デュアルスタック (IPv4 および IPv6) 接続をサポートしています。デュアルスタック VPC エンドポイントを作成すると、IPv4 プライベート IP アドレスと IPv6

プライベート IP アドレスの両方でトラフィックが自動的に処理されます。エンドポイントは、クライアントのネットワーク設定とエンドポイントの機能に基づいて、適切な IP バージョンを使用してトラフィックをルーティングします。

既存の IPv4-only VPC エンドポイントがあり、デュアルスタックエンドポイントに移行する場合は、既存のエンドポイントを変更してデュアルスタック接続をサポートするか、新しいデュアルスタックエンドポイントを作成できます。エンドポイントを作成または変更するときは、VPC とサブネットが使用する IP バージョンをサポートしていることを確認してください。デュアルスタックエンドポイントを作成すると、エンドポイントは IPv4 と IPv6 の両方をサポートします。

Amazon EC2 インスタンスでホストされる Amazon ECS タスクの場合、Amazon ECR エンドポイントと Amazon S3 ゲートウェイエンドポイントの両方が必要です。

プラットフォームバージョン `1.4.0` 以降を使用する Fargate でホストされる Amazon ECS タスクの場合、Amazon ECR VPC エンドポイントと Amazon S3 ゲートウェイエンドポイントの両方が必要です。

プラットフォームバージョン `1.3.0` 以前を使用する Fargate でホストされる Amazon ECS タスクの場合、必要なのは com.amazonaws.***region***.ecr.dkr Amazon ECR VPC エンドポイントと Amazon S3 ゲートウェイエンドポイントのみです。

> **Note**
>
> エンドポイントが作成される順序は重要ではありません。

com.amazonaws.***region***.ecr.dkr

このエンドポイントは、Docker Registry API に使用されます。`push` や `pull` などの Docker クライアントコマンドでは、このエンドポイントが使用されます。

このエンドポイントを作成する際に、プライベート DNS ホスト名を有効にする必要があります。 これを行うには、VPC エンドポイントを作成するときに、Amazon VPC コンソールで [プライベート DNS 名を有効にする] オプションが選択されていることを確認します。

FIPS 140-3 準拠の接続の場合は、FIPS エンドポイント名 com.amazonaws.***region***.ecr-fips.dkr を使用します。

com.amazonaws.*region*.ecr.api

> **Note**
>
> 指定された*#####*は、米国東部 (オハイオ) リージョンなど、Amazon ECR でサポートされている AWS リージョン`us-east-2`のリージョン識別子を表します。

FIPS 140-3 準拠の接続には、FIPS エンドポイント名 com.amazonaws.*region*.ecr-fips.dkr と com.amazonaws.*region*.ecr-fips.api を使用します。

このエンドポイントは、Amazon ECR API への呼び出しに使用されます。`DescribeImages` や `CreateRepository` などの API アクションは、このエンドポイントに移動します。

このエンドポイントを作成すると、プライベート DNS ホスト名を有効にするオプションが使用可能になります。VPC エンドポイントの作成時に VPC コンソールで [プライベート DNS 名を有効にする] を選択して、この設定名を有効にします。VPC エンドポイントのプライベート DNS ホスト名を有効にする場合は、SDK または を最新バージョン AWS CLI に更新して、SDK または AWS CLI を使用するときにエンドポイント URL を指定する必要はありません。

FIPS 140-3 準拠の接続の場合は、FIPS エンドポイント名 com.amazonaws.*region*.ecr-fips.api を使用します。

プライベート DNS ホスト名を有効にし、2019 年 1 月 24 日より前にリリースされた SDK または AWS CLI バージョンを使用している場合は、 `--endpoint-url`パラメータを使用してインターフェイスエンドポイントを指定する必要があります。次の例は、エンドポイント URL の形式を示しています。

```
aws ecr create-repository --repository-name name --endpoint-url https://
api.ecr.region.amazonaws.com
```

VPC エンドポイントでプライベート DNS ホスト名を有効にしない場合は、インターフェイスエンドポイントで VPC エンドポイント ID を指定する `--endpoint-url` パラメータを使用する必要があります。次の例は、エンドポイント URL の形式を示しています。

```
aws ecr create-repository --repository-name name --endpoint-url
 https://VPC_endpoint_ID.api.ecr.region.vpce.amazonaws.com
```

FIPS 140-3 準拠の接続の場合は、FIPS エンドポイント URL を使用します。

```
aws ecr create-repository --repository-name name --endpoint-url https://api.ecr-fips.region.amazonaws.com
```

Amazon S3 ゲートウェイエンドポイントを作成する

Amazon ECS タスクで Amazon ECR からプライベートイメージをプルするには、Amazon S3 のゲートウェイエンドポイントを作成する必要があります。Amazon ECR は Amazon S3 を使用してイメージレイヤーを保存するため、ゲートウェイエンドポイントが必要です。Amazon ECR からイメージをダウンロードするコンテナは、Amazon ECR にアクセスしてイメージマニフェストを取得してから Amazon S3 にアクセスして実際のイメージレイヤーをダウンロードする必要があります。各 Docker イメージのレイヤーを含む Amazon S3 バケットの Amazon リソースネーム (ARN) を以下に示します。

```
arn:aws:s3:::prod-region-starport-layer-bucket/*
```

> **Note**
>
> Amazon ECR のデュアルスタック VPC エンドポイントを作成する場合は、デュアルスタックの Amazon S3 Gateway またはインターフェイスエンドポイントも作成する必要があります。詳細については、S3 ドキュメントを参照してください。

Amazon ECR に以下の Amazon S3 ゲートウェイエンドポイントを作成するには、Amazon VPC ユーザーガイドのゲートウェイエンドポイントの作成手順を使用します。エンドポイントを作成するときは、必ず VPC のルートテーブルを選択してください。

com.amazonaws.*region*.s3

Amazon S3 ゲートウェイエンドポイントは IAM ポリシードキュメントを使用してサービスへのアクセスを制限します。 このポリシーには、タスクの IAM ロールまたはその他の IAM ユーザーポリシーに設定された制限が引き続き優先して適用されるので、フルアクセスポリシーを使用できます。Amazon S3 バケットアクセスを Amazon ECR を使用するための最小限のアクセス許可に制限する場合は、「Amazon ECR の最小 Amazon S3 バケットアクセス許可」を参照してください。

Amazon ECR の最小 Amazon S3 バケットアクセス許可

Amazon S3 ゲートウェイエンドポイントは IAM ポリシードキュメントを使用してサービスへのアクセスを制限します。 Amazon ECR に最低限の Amazon S3 バケットアクセス許可のみを許可するには、エンドポイントの IAM ポリシードキュメントを作成するときに Amazon ECR が使用する Amazon S3 バケットへのアクセスを制限します。

次の表は、Amazon ECR に必要な Amazon S3 バケットポリシーアクセス許可を示しています。

アクセス許可	説明
`arn:aws:s3:::prod-` *`region`*`-starport-layer-bucket/*`	各 Docker イメージのレイヤーを含む Amazon S3 バケットへのアクセスを提供します。米国東部 (オハイオ) リージョンの `us-east-2` のように、Amazon ECR でサポートされている AWS リージョンのリージョン識別子を表します。

例

以下の例は、Amazon ECR オペレーションに必要な Amazon S3 バケットへのアクセスを提供する方法を示しています。

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::prod-region-starport-layer-bucket/*"]
    }
  ]
}
```

CloudWatch Logs エンドポイントを作成する

Fargate 起動タイプを使用し、インターネットゲートウェイなしで VPC を使用する一方で、awslogs ログドライバーを使用してログ情報を CloudWatch に送信する Amazon ECS タスクの場合、CloudWatch Logs の com.amazonaws.***region***.logs インターフェイス VVPC エンドポイントを作成する必要があります。詳細については、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。

Amazon ECR VPC エンドポイントのエンドポイントポリシーを作成する

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国際機械技術者協会 (IAM) の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、 は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をア AWS タッチします。エンドポイントポリシーは、 ユーザー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。

1 つの IAM リソースポリシーを作成し、両方の Amazon ECR VPC エンドポイントにアタッチすることをお勧めします。

Amazon ECR API のエンドポイントポリシーの例を次に示します。このポリシーは、特定の IAM ロールが Amazon ECR からイメージをプルできるようにします。

```
{
 "Statement": [{
  "Sid": "AllowPull",
  "Principal": {
   "AWS": "arn:aws:iam::1234567890:role/role_name"
  },
  "Action": [
   "ecr:BatchGetImage",
   "ecr:GetDownloadUrlForLayer",
                  "ecr:GetAuthorizationToken"
  ],
  "Effect": "Allow",
  "Resource": "*"
 }]
}
```

次のエンドポイントポリシーの例では、指定されたリポジトリが削除されないようにしています。

```
{
 "Statement": [{
   "Sid": "AllowAll",
   "Principal": "*",
   "Action": "*",
   "Effect": "Allow",
   "Resource": "*"
  },
  {
   "Sid": "PreventDelete",
   "Principal": "*",
   "Action": "ecr:DeleteRepository",
   "Effect": "Deny",
   "Resource": "arn:aws:ecr:region:1234567890:repository/repository_name"
  }
 ]
}
```

次のエンドポイントポリシーの例では、前述の 2 つの例を 1 つのポリシーにまとめています。

```
{
 "Statement": [{
   "Sid": "AllowAll",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "*",
   "Resource": "*"
  },
  {
   "Sid": "PreventDelete",
   "Effect": "Deny",
   "Principal": "*",
   "Action": "ecr:DeleteRepository",
   "Resource": "arn:aws:ecr:region:1234567890:repository/repository_name"
  },
  {
   "Sid": "AllowPull",
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::1234567890:role/role_name"
   },
```

```
    "Action": [
     "ecr:BatchGetImage",
     "ecr:GetDownloadUrlForLayer",
                           "ecr:GetAuthorizationToken"
    ],
    "Resource": "*"
   }
  ]
}
```

Amazon ECR 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. Amazon ECR の VPC エンドポイントをまだ作成していない場合は、「Amazon ECR の VPC エンドポイントを作成する」を参照してください。
4. ポリシーを追加する Amazon ECR VPC エンドポイントを選択し、画面の下部にある [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。
6. [保存] を選択してポリシーを保存します。

共有サブネット

自分と共有されているサブネットで VPC エンドポイントを作成、説明、変更、または削除することはできません。ただし、VPC エンドポイントを使用することはできます。

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、 AWS では、アカウント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で、すべてのサービスのデータを保護するために役立つツールを提供しています。

リソースポリシー内では aws:SourceArn または aws:SourceAccount のグローバル条件コンテキストキーを使用して、Amazon ECR が別のサービスに付与する、リソースへのアクセス許可を制限することをお勧めします。クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、aws:SourceArn を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、aws:SourceAccount を使用します。

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*:*region*:*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、両方のグローバル条件コンテキストキーを使用して、アクセス許可を制限する必要があります。

aws:SourceArn の値は ResourceDescription である必要があります。

次の例は、Amazon ECR リポジトリポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、そのサービスとの統合に必要な Amazon ECR API アクション AWS CodeBuild へのアクセスを許可し、混乱した代理問題を防ぐ方法を示しています。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CodeBuildAccess",
         "Effect":"Allow",
         "Principal":{
            "Service":"codebuild.amazonaws.com"
         },
         "Action":[
            "ecr:BatchGetImage",
            "ecr:GetDownloadUrlForLayer"
         ],
         "Resource": "*",
         "Condition":{
            "ArnLike":{
               "aws:SourceArn":"arn:aws:codebuild:us-
east-1:123456789012:project/project-name"
```

```
            },
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            }
         }
      }
   ]
}
```

Amazon ECR のモニタリング

Amazon CloudWatch を使用して Amazon ECR API の使用状況をモニタリングすることで、Amazon ECR から raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに加工することができます。これらの統計情報は 2 週間記録されるため、履歴情報にアクセスして API の使用状況を把握できます。Amazon ECR のメトリクスデータは 1 分間隔で自動的に CloudWatch に送信されます。CloudWatch の詳細については、Amazon CloudWatch ユーザーガイドを参照してください。

Amazon ECR には、認可、イメージプッシュ、およびイメージプルアクションの API の使用状況に基づいたメトリクスが用意されています。

モニタリングは、Amazon ECR と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションを構成するリソースからモニタリングデータを収集することをお勧めします。ただし、Amazon ECR のモニタリングを開始する前に、以下の質問に対する回答を反映したモニタリング計画を作成する必要があります。

- モニタリングの目的は何ですか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまなタイミングと負荷条件でパフォーマンスを測定することにより、お客様の環境で通常の Amazon ECR のパフォーマンスのベースラインを確定します。Amazon ECR のモニタリングでは、過去のモニタリングデータを保存し、新しいパフォーマンスデータと比較することで、パフォーマンスの通常パターンと異常パターンを特定し、問題に対処する方法を考案できます。

トピック

- サービスクォータの可視化とアラームの設定
- Amazon ECR 使用状況メトリクス
- Amazon ECR 使用状況レポート
- Amazon ECR リポジトリメトリクス

- Amazon ECR イベントと EventBridge
- を使用した Amazon ECR アクションの ログ記録 AWS CloudTrail

サービスクォータの可視化とアラームの設定

CloudWatch コンソールを使用して、サービスクォータを可視化し、サービスクォータと照らして現在の使用状況を確認できます。クォータに近づいたときに通知されるようにアラームを設定することもできます。

サービスクォータを可視化し、オプションでアラームを設定するには

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. [すべてのメトリクス] タブで [使用状況] を選択し、[AWS リソース] を選択します。

   サービスクォータ使用状況メトリクスのリストが表示されます。
4. いずれかのメトリクスの横にあるチェックボックスをオンにします。

   グラフには、その AWS リソースの現在の使用状況が表示されます。
5. サービスクォータをグラフに追加するには、次の手順を実行します。

   a. [グラフ化したメトリクス] タブを選択します。

   b. [Math expression (数式)]、[Start with an empty expression (空の式で開始)] の順に選択します。次に、新しい行の [Details (詳細)] に「**`SERVICE_QUOTA(m1)`**」と入力します。

      グラフに新しい行が追加され、メトリクスで表されるリソースのサービスクォータが表示されます。
6. 現在の使用状況をクォータの割合として表示するには、新しい式を追加するか、現在の [SERVICE_QUOTA] 式を変更します。新しい式には、**`m1/60/SERVICE_QUOTA(m1)*100`** を使用します。
7. （オプション）サービスクォータに近づいた場合に通知するアラームを設定するには、次の手順を実行します。

   a. **`m1/60/SERVICE_QUOTA(m1)*100`** 行の [Actions (アクション)] で、アラームアイコンを選択します。それはベルのように見えます。

      アラーム作成ページが表示されます。

b. [Conditions (条件)] で、[Threshold type (しきい値の種類)] が [Static (静的)] で、[Whenever Expression1 is (式 1)] が [Greater (大きい)] に設定されていることを確認します。[than (以上)] に「**80**」と入力します。これにより、使用量がクォータの 80% を超えたときに ALARM 状態になるアラームが作成されます。

c. [次へ] を選択します。

d. 次のページで、Amazon SNS トピックを選択するか、新しいトピックを作成します。このトピックは、アラームが ALARM 状態になったときに通知されます。次いで、[次へ] を選択します。

e. 次のページで、アラームの名前と説明を入力し、[Next] (次へ) を選択します。

f. [アラームの作成] を選択します。

Amazon ECR 使用状況メトリクス

CloudWatch 使用状況メトリクスを使用して、アカウントのリソースの使用状況を把握できます。これらのメトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。

Amazon ECR 使用状況メトリクスは、 AWS サービスクォータに対応しています。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。Amazon ECR のサービスのクォータの詳細については、「Amazon ECR のサービスクォータ」を参照してください。

Amazon ECR は、`AWS/Usage` 名前空間に以下のメトリクスを公開します。

メトリクス	説明
`CallCount`	アカウントからの API アクションの呼び出し回数。リソースはメトリクスに関連付けられたディメンションによって定義されます。 このメトリクスの最も便利な統計は `SUM` であり、定義した期間中のすべての寄稿者からの値の合計を表します。
`ResourceCount`	アカウントの指定されたリソースの数。リソースはメトリクスに関連付けられたディメンションによって定義されます。 このメトリクスの最も有用な統計は です。これは`MAXIMUM`、5 分間に使用されるリソースの最大数を表します。

次のディメンションは、Amazon ECR によって公開される API 使用状況メトリクスを絞り込むために使用されます。

ディメンション	説明
Service	リソースを含む AWS サービスの名前。Amazon ECR 使用状況メトリクスの場合、このディメンションの値は ECR です。
Type	報告されるエンティティタイプ。現在、Amazon ECR API 使用状況メトリクスの有効な値は のみですAPI。
Resource	実行中のリソースタイプ。現在、Amazon ECR は以下の API アクションの API の使用状況に関する情報を返します。 • GetAuthorizationToken • BatchCheckLayerAvailability • InitiateLayerUpload • UploadLayerPart • CompleteLayerUpload • PutImage • BatchGetImage • GetDownloadUrlForLayer
Class	追跡されているリソースのクラス。現在、Amazon ECR はクラスディメンションを使用していません。

次のディメンションは、Amazon ECR によって公開されるリソース使用状況メトリクスを絞り込むために使用されます。

ディメンション	説明
Service	リソースを含む AWS サービスの名前。Amazon ECR 使用状況メトリクスの場合、このディメンションの値は ECR です。
Type	報告されるエンティティタイプ。現在、Amazon ECR リソース使用状況メトリクスの有効な値は のみですRESOURCE。

ディメンション	説明
Resource	実行中のリソースタイプ。現在、Amazon ECR は、次のメトリクスのリソース使用状況に関する情報を返します。 • RepositoryCount • ImagesPerRepositoryCount
ResourceId	使用量が発生したリソースの識別子。現在、ResourceId はにのみ関連ImagesPerRepositoryCount し、その値は repository/your_repository_name としてフォーマットされています。たとえば、「repository/my-repo」は、リポジトリ内のイメージの数を「my-repo」という名前で返します。

Amazon ECR 使用状況レポート

AWS には、Amazon ECR リソースのコストと使用状況を分析できる Cost Explorer と呼ばれる無料のレポートツールが用意されています。

Cost Explorer を使用して、使用状況とコストのグラフを表示します。過去 13 か月からデータを表示でき、また次の 3 か月間にどのくらい使用する可能性があるかを予測します。時間の経過に伴う AWS リソースの使用量パターンを確認して、さらに照会が必要な分野を識別し、コストの把握に役立つ傾向を確認するには、Cost Explorer を使用します。データの時間範囲を指定したり、時間データを日または月ごとに表示したりもできます。

コストおよび使用状況レポートの計測データには、すべての Amazon ECR リポジトリの使用状況が示されます。詳細については、「請求用のリソースにタグを付ける」を参照してください。

AWS コストと使用状況レポートの作成の詳細については、「 AWS Billing ユーザーガイド」のAWS「コストと使用状況レポート」を参照してください。

Amazon ECR リポジトリメトリクス

Amazon ECR はリポジトリのプルカウントメトリクスを Amazon CloudWatch に送信します。Amazon ECR のメトリクスデータは 1 分間隔で自動的に CloudWatch に送信されます。CloudWatch の詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

トピック

- CloudWatch メトリクスの有効化
- 使用できるメトリクスとディメンション
- CloudWatch コンソールを使用して Amazon ECR メトリクスを表示するには

CloudWatch メトリクスの有効化

Amazon ECR は、すべてのリポジトリのリポジトリメトリクスを自動的に送信します。手動の手順を実行する必要はありません。

使用できるメトリクスとディメンション

以下のセクションでは、 Amazon ECR が Amazon CloudWatchに送信するメトリクスとディメンションを一覧表示しています。

Amazon ECR メトリクス

Amazon ECR には、リポジトリをモニタリングするためのメトリクスが用意されています。プルカウントを測定できます。

AWS/ECR 名前空間には、次のメトリクスが含まれます。

`RepositoryPullCount`

リポジトリ内のイメージのプルの総数。

有効なディメンション: `RepositoryName`。

有効な統計: 平均、最小、最大、合計、サンプル数。最も有用な統計は Sum です。

単位: 整数。

Amazon ECR メトリクスのディメンション

Amazon ECR メトリクスは AWS/ECR 名前空間を使用し、以下のディメンションのメトリクスを提供しています。

`RepositoryName`

このディメンションは、指定したリポジトリのすべてのコンテナイメージに対してリクエストしたデータをフィルターします。

CloudWatch コンソールを使用して Amazon ECR メトリクスを表示するには

CloudWatch コンソールで Amazon ECR レポジトリメトリクスを表示できます。CloudWatch コンソールは、リポジトリの詳細でカスタマイズ可能な表示を提供します。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

Amazon ECR イベントと EventBridge

Amazon EventBridge を使用すると、 AWS サービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 AWS のサービスからのイベントは、ほぼリアルタイムに EventBridge に提供されます。簡単なルールを作成し、ルールで対象とするイベントを指定し、イベントがルールに一致した場合に自動的に実行するアクションを含めることができます。自動的にトリガーできるオペレーションには、以下が含まれます。

- CloudWatch Logs のロググループへのイベントの追加
- AWS Lambda 関数の呼び出し
- Amazon EC2 Run Command の呼び出し
- Amazon Kinesis Data Streams へのイベントの中継
- AWS Step Functions ステートマシンのアクティブ化
- Amazon SNS トピックまたは Amazon SQS キューの通知

詳細については、Amazon EventBridge ユーザーガイドの「Amazon EventBridge の開始方法」を参照してください 。

Amazon ECR のサンプルイベント

以下に、Amazon ECR のイベントの例を示します。イベントは、ベストエフォートベースで出力されます。

完了したイメージプッシュのイベント

各イメージプッシュが完了すると、以下のイベントが送信されます。詳細については、「Amazon ECR プライベートリポジトリへの Docker イメージのプッシュ」を参照してください。

```
{
    "version": "0",
    "id": "13cde686-328b-6117-af20-0e5566167482",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-11-16T01:54:34Z",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "result": "SUCCESS",
        "repository-name": "my-repository-name",
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "action-type": "PUSH",
        "image-tag": "latest"
    }
}
```

プルスルーキャッシュアクションのイベント

プルスルーキャッシュアクションが試行されると、次のイベントが送信されます。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリの同期」を参照してください。

```
{
    "version": "0",
    "id": "85fc3613-e913-7fc4-a80c-a3753e4aa9ae",
    "detail-type": "ECR Pull Through Cache Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2023-02-29T02:36:48Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecr:us-west-2:123456789012:repository/docker-hub/alpine"
    ],
    "detail": {
        "rule-version": "1",
        "sync-status": "SUCCESS",
```

```
        "ecr-repository-prefix": "docker-hub",
        "repository-name": "docker-hub/alpine",
        "upstream-registry-url": "public.ecr.aws",
        "image-tag": "3.17.2",
        "image-digest":
 "sha256:4aa08ef415aecc80814cb42fa41b658480779d80c77ab15EXAMPLE",
    }
}
```

完了したイメージスキャンのイベント (ベーシックスキャン)

レジストリでベーシックスキャンを有効にすると、各イメージスキャンが完了した際、以下のイベントが送信されます。finding-severity-counts パラメータは、重要度レベルが存在する場合にのみ、その値を返します。たとえば、イメージに CRITICAL レベルの結果が含まれていない場合、重要度のカウントは返されません。詳細については、「Amazon ECR でイメージをスキャンして OS 脆弱性がないか調べる」を参照してください。

> **Note**
>
> 拡張スキャンが有効なときに Amazon Inspector によって発行されるイベントの詳細については、Amazon ECR の拡張スキャンについて送信される EventBridge イベント を参照してください。

```
{
    "version": "0",
    "id": "85fc3613-e913-7fc4-a80c-a3753e4aa9ae",
    "detail-type": "ECR Image Scan",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-10-29T02:36:48Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecr:us-east-1:123456789012:repository/my-repository-name"
    ],
    "detail": {
        "scan-status": "COMPLETE",
        "repository-name": "my-repository-name",
        "finding-severity-counts": {
        "CRITICAL": 10,
        "MEDIUM": 9
```

```
      },
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "image-tags": []
    }
}
```

拡張スキャンが有効なリソースについての変更通知のイベント (拡張スキャン)

レジストリで拡張スキャンが有効になっている場合、拡張スキャンが有効になっているリソースで変更があった際、Amazon ECR から次のイベントが送信されます。これには、新しいリポジトリの作成、リポジトリのスキャン頻度の変更、または拡張スキャンが有効になっているリポジトリでのイメージの作成または削除が含まれます。詳しくは、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。

```
{
 "version": "0",
 "id": "0c18352a-a4d4-6853-ef53-0ab8638973bf",
 "detail-type": "ECR Scan Resource Change",
 "source": "aws.ecr",
 "account": "123456789012",
 "time": "2021-10-14T20:53:46Z",
 "region": "us-east-1",
 "resources": [],
 "detail": {
  "action-type": "SCAN_FREQUENCY_CHANGE",
  "repositories": [{
    "repository-name": "repository-1",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-1",
    "scan-frequency": "SCAN_ON_PUSH",
    "previous-scan-frequency": "MANUAL"
   },
   {
    "repository-name": "repository-2",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-2",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   },
   {
    "repository-name": "repository-3",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-3",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
```

```
    }
   ],
   "resource-type": "REPOSITORY",
   "scan-type": "ENHANCED"
 }
}
```

イメージ削除のイベント

イメージが削除されると、以下のイベントが送信されます。詳細については、「Amazon ECR でのイメージの削除」を参照してください。

```
{
    "version": "0",
    "id": "dd3b46cb-2c74-f49e-393b-28286b67279d",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-11-16T02:01:05Z",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "result": "SUCCESS",
        "repository-name": "my-repository-name",
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "action-type": "DELETE",
        "image-tag": "latest"
    }
}
```

イメージアーカイブアクションのイベント

イメージがアーカイブされると、次のイベントが送信されます。target-storage-class フィールドは に設定されますARCHIVE。イベントには、アーカイブされるコンテンツのタイプを識別するためのマニフェストメディアタイプとアーティファクトメディアタイプが含まれます。

```
{
    "version": "0",
    "id": "4f5ec4d5-4de4-7aad-a046-EXAMPLE",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
```

```
    "account": "123456789012",
    "time": "2019-08-06T00:58:09Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "action-type": "UPDATE_STORAGE_CLASS",
        "target-storage-class": "ARCHIVE",
        "image-digest":
 "sha256:f98d67af8e53a536502bfc600de3266556b06ed635a32d60aa7a5fe6d7e609d7",
        "repository-name": "ubuntu",
        "result": "SUCCESS",
        "manifest-media-type": "application/vnd.oci.image.manifest.v1+json",
        "artifact-media-type": "application/vnd.oci.image.config.v1+json"
    }
}
```

イメージ復元アクションのイベント

アーカイブされたイメージが復元されると、次のイベントが送信されます。target-storage-class フィールドは に設定されますSTANDARD。イベントには、イメージが最後に復元された日時を示すlast-activated-atフィールドが含まれます。

```
{
    "version": "0",
    "id": "7b8fc5e6-5ef5-8bbe-b157-EXAMPLE",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-08-06T01:15:22Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "action-type": "UPDATE_STORAGE_CLASS",
        "target-storage-class": "STANDARD",
        "image-digest":
 "sha256:f98d67af8e53a536502bfc600de3266556b06ed635a32d60aa7a5fe6d7e609d7",
        "repository-name": "ubuntu",
        "result": "SUCCESS",
        "manifest-media-type": "application/vnd.oci.image.manifest.v1+json",
        "artifact-media-type": "application/vnd.oci.image.config.v1+json",
        "last-activated-at": "2025-10-10T19:13:02.74Z"
    }
}
```

リファラー復元アクションのイベント

アーカイブされたリファラ (SBOM、署名、認証などの参照アーティファクト) が復元されると、次のイベントが送信されます。detail-type は、通常のイメージアクションと区別ECR Referrer Actionするためのものです。フィールドmanifest-media-typeと artifact-media-typeフィールドは、復元されるリファラの特定のタイプを識別します。この例では、SBOM アーティファクトが復元されています。

```
{
    "version": "0",
    "id": "8c9gd6f7-6fg6-9ccf-c268-EXAMPLE",
    "detail-type": "ECR Referrer Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-08-06T01:20:45Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "action-type": "UPDATE_STORAGE_CLASS",
        "target-storage-class": "STANDARD",
        "image-digest":
 "sha256:f98d67af8e53a536502bfc600de3266556b06ed635a32d60aa7a5fe6d7e609d7",
        "repository-name": "sbom",
        "result": "SUCCESS",
        "manifest-media-type": "application/vnd.cncf.oras.artifact.manifest.v1+json",
        "artifact-media-type": "text/sbom+json",
        "last-activated-at": "2025-10-10T19:13:02.74Z"
    }
}
```

完了したイメージレプリケーションに関するイベント

各イメージレプリケーションが完了すると、次のイベントが送信されます。詳細については、「Amazon ECR でのプライベートイメージレプリケーション」を参照してください。

```
{
  "version": "0",
  "id": "c8b133b1-6029-ee73-e2a1-4f466b8ba999",
  "detail-type": "ECR Replication Action",
  "source": "aws.ecr",
  "account": "123456789012",
  "time": "2024-05-08T20:44:54Z",
```

```
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecr:us-east-1:123456789012:repository/docker-hub/alpine"
  ],
  "detail": {
    "result": "SUCCESS",
    "repository-name": "docker-hub/alpine",
    "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
    "source-account": "123456789012",
    "action-type": "REPLICATE",
    "source-region": "us-west-2",
    "image-tag": "3.17.2"
  }
}
```

失敗したイメージレプリケーションに関するイベント

イメージレプリケーションが失敗すると、次のイベントが送信されます。result フィールドには FAILED が含まれ、イベントの詳細に追加のエラー情報が含まれる場合があります。

```
{
  "version": "0",
  "id": "d9c244c2-7130-ff84-f3b2-5g577c9cb000",
  "detail-type": "ECR Replication Action",
  "source": "aws.ecr",
  "account": "123456789012",
  "time": "2024-05-08T20:45:12Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecr:us-east-1:123456789012:repository/my-app"
  ],
  "detail": {
    "result": "FAILED",
    "repository-name": "my-app",
    "image-digest":
 "sha256:8g6c3751gf7gc5g47603ege4511d5a80ead5g90f5893543f1489bde2345",
    "source-account": "123456789012",
    "action-type": "REPLICATE",
    "source-region": "us-west-2",
    "image-tag": "latest"
  }
}
```

を使用した Amazon ECR アクションのログ記録 AWS CloudTrail

Amazon ECR は AWS CloudTrail、Amazon ECR のユーザー、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています。CloudTrail は、次の Amazon ECR アクションをイベントとしてキャプチャします。

- Amazon ECR コンソールからのコールを始めとするすべての API コール
- リポジトリの暗号化設定によって実行されたすべてのアクション
- 成功アクションと失敗アクションの両方を含む、ライフサイクルポリシールールによって実行されるすべてのアクション

> **Important**
>
> 個々の CloudTrail イベントのサイズ制限により、10 個以上のイメージが期限切れになるライフサイクルポリシーアクションの場合、Amazon ECR が複数のイベントを CloudTrail に送信します。また、Amazon ECR には、イメージごとに最大 100 個のタグが含まれます。

証跡を作成すると、Amazon ECR のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。この情報を使用して、Amazon ECR に対して行われたリクエスト、リクエストの作成元の IP アドレス、リクエストの実行者、リクエストの実行日時などの詳細を特定できます。

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

CloudTrail の Amazon ECR 情報

CloudTrail は、 AWS アカウントの作成時にアカウントで有効になります。Amazon ECR でアクティビティが発生すると、そのアクティビティは [Event history (イベント履歴)] の他の AWS のサービスのイベントとともに CloudTrail イベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

Amazon ECR のイベントなど、 AWS アカウントのイベントの継続的な記録については、証跡を作成します。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。コンソールで証跡を作成する場合、証跡を単一のリージョンまたはすべてのリージョンに適用できます。

証跡は AWS パーティション内のイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、その他の AWS サービスを設定して、CloudTrail ログで収集したデータを分析し、それに基づく対応を行うことができます。詳細については、以下を参照してください。

- AWS アカウントの証跡の作成
- AWS CloudTrail ログとの サービス統合
- 「CloudTrail の Amazon SNS 通知の設定」
- CloudTrail ログファイルを複数のリージョンから受け取る、複数のアカウントから CloudTrail ログファイルを受け取る

すべてのAmazon ECR API アクションは、CloudTrail が記録し、Amazon Elastic Container Registry API リファレンスで説明されています。一般的なタスクを実行すると、そのタスクに含まれる各 API アクションのセクションが CloudTrail ログファイルに生成されます。たとえば、リポジトリを作成すると、`GetAuthorizationToken`、`CreateRepository`、`SetRepositoryPolicy` のセクションが CloudTrail ログファイルに生成されます。イメージをリポジトリ、、、`InitiateLayerUploadUploadLayerPartCompleteLayerUpload`、 `PutImage`にプッシュすると、BLOB マウントが有効になっている場合、`MountLayer`セクションが生成されます。イメージをプルすると、`GetDownloadUrlForLayer` と `BatchGetImage` のセクションが生成されます。アーカイブまたは復元すると、イメージ`UpdateImageStorageClass`セクションが生成されます。OCI 1.1 仕様をサポートする OCI クライアントが、Referrers API を使用してイメージのリファラーまたはリファレンスアーティファクトのリストを取得すると、`ListImageReferrers` CloudTrail イベントが発行されます。これらの一般的なタスクの例については、「CloudTrail ログエントリの例」を参照してください。

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。

- リクエストが、ルートと ユーザー認証情報のどちらを使用して送信されたか
- リクエストの送信に使用された一時的なセキュリティ認証情報に、ロールとフェデレーテッドユーザーのどちらが使用されたか
- 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。

Amazon ECR ログファイルエントリの理解

「トレイル」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 のログファイルには、ログエントリが 1 つ以上あります。イベントは任意の発生元からの 1 つ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、リクエストの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、公開 API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

CloudTrail ログエントリの例

いくつかの一般的な Amazon ECR タスクの CloudTrail ログエントリの例を次に示します。

これらの例は、読みやすくするために書式設定されています。CloudTrail ログファイルでは、すべてのエントリとイベントが単一の行に連結されています。さらに、この例は 1 つの Amazon ECR エントリに限定しています。実際の CloudTrail ログファイルには、複数の AWS サービスからのエントリとイベントが表示されます。

> **Important**
>
> sourceIPAddress はリクエストが行われた IP アドレスです。サービスコンソールから行われたアクションの場合、報告されるアドレスは、コンソールウェブサーバーではなく、基礎となるリソースのものです。のサービスの場合 AWS、DNS 名のみが表示されます。クライアントソース IP が AWS サービス DNS 名に置き換えられている場合でも、認証の評価はその IP を使用して行われます。

トピック

- 例: リポジトリの作成アクション
- 例: Amazon ECR リポジトリの作成時の AWS KMS CreateGrant API アクション
- 例: イメージプッシュアクション
- 例: イメージプルアクション
- 例: イメージライフサイクルポリシーアクション
- 例: イメージアーカイブアクション
- 例: イメージ復元アクション
- 例: イメージリファラーアクション

例: リポジトリの作成アクション

以下の例は、CreateRepository アクションを示す CloudTrail ログエントリです。


```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-07-11T21:54:07Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            }
        }
    },
    "eventTime": "2018-07-11T22:17:43Z",
    "eventSource": "ecr.amazonaws.com",
    "eventName": "CreateRepository",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "repositoryName": "testrepo"
    },
    "responseElements": {
        "repository": {
            "repositoryArn": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
            "repositoryName": "testrepo",
            "repositoryUri": "123456789012.dkr.ecr.us-east-2.amazonaws.com/testrepo",
            "createdAt": "Jul 11, 2018 10:17:44 PM",
            "registryId": "123456789012"
        }
    },
```

```
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "resources": [
        {
            "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
            "accountId": "123456789012"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

例: Amazon ECR リポジトリの作成時の AWS KMS **CreateGrant** API アクション

次の例は、KMS 暗号化を有効にして Amazon ECR リポジトリを作成するときの AWS KMS CreateGrantアクションを示す CloudTrail ログエントリを示しています。KMS 暗号化を有効にして作成されたリポジトリごとに、2 つの CreateGrant ログエントリが CloudTrail に表示されます。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAIEP6W46J43IG7LXAQ",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "sessionIssuer": {

            },
            "webIdFederationData": {

            },
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-06-10T19:22:10Z"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-06-10T19:22:10Z",
```

```
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "keyId": "4b55e5bf-39c8-41ad-b589-18464af7758a",
        "granteePrincipal": "ecr.us-west-2.amazonaws.com",
        "operations": [
            "GenerateDataKey",
            "Decrypt"
        ],
        "retiringPrincipal": "ecr.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecr:arn": "arn:aws:ecr:us-west-2:123456789012:repository/testrepo"
            }
        }
    },
    "responseElements": {
        "grantId": "3636af9adfee1accb67b83941087dcd45e7fadc4e74ff0103bb338422b5055f3"
    },
    "requestID": "047b7dea-b56b-4013-87e9-a089f0f6602b",
    "eventID": "af4c9573-c56a-4886-baca-a77526544469",
    "readOnly": false,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:123456789012:key/4b55e5bf-39c8-41ad-
b589-18464af7758a"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


例: イメージプッシュアクション

以下の例は、PutImage アクションを使用するイメージのプッシュの CloudTrail ログエントリを示しています。

> **Note**
>
> イメージをプッシュすると、InitiateLayerUpload、UploadLayerPart、および CompleteLayerUpload の参照も CloudTrail ログに表示されます。


```
{
    "eventVersion": "1.04",
    "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
    "arn": "arn:aws:sts::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T16:45:00Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "PutImage",
 "awsRegion": "us-east-2",
 "sourceIPAddress": "AWS Internal",
 "userAgent": "AWS Internal",
 "requestParameters": {
  "repositoryName": "testrepo",
  "imageTag": "latest",
  "registryId": "123456789012",
  "imageManifest": "{\n   \"schemaVersion\": 2,\n   \"mediaType\": \"application/
vnd.docker.distribution.manifest.v2+json\",\n   \"config\": {\n      \"mediaType\":
 \"application/vnd.docker.container.image.v1+json\",\n      \"size\": 5543,\n
 \"digest\": \"sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a
\"\n   },\n   \"layers\": [\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 43252507,\n
 \"digest\": \"sha256:3b37166ec61459e76e33282dda08f2a9cd698ca7e3d6bc44e6a6e7580cdeff8e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 846,\n         \"digest
\": \"sha256:504facff238fde83f1ca8f9f54520b4219c5b8f80be9616ddc52d31448a044bd
```

```
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 615,\n         \"digest
\": \"sha256:ebbcacd28e101968415b0c812b2d2dc60f969e36b0b08c073bf796e12b1bb449\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 850,\n         \"digest
\": \"sha256:c7fb3351ecad291a88b92b600037e2435c84a347683d540042086fe72c902b8a
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 168,\n         \"digest\":
 \"sha256:2e3debadcbf7e542e2aefbce1b64a358b1931fb403b3e4aeca27cb4d809d56c2\"\n      },
\n      {\n         \"mediaType\": \"application/vnd.docker.image.rootfs.diff.tar.gzip
\",\n         \"size\": 37720774,\n         \"digest\":
 \"sha256:f8c9f51ad524d8ae9bf4db69cd3e720ba92373ec265f5c390ffb21bb0c277941\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 30432107,\n
 \"digest\": \"sha256:813a50b13f61cf1f8d25f19fa96ad3aa5b552896c83e86ce413b48b091d7f01b
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 197,\n         \"digest
\": \"sha256:7ab043301a6187ea3293d80b30ba06c7bf1a0c3cd4c43d10353b31bc0cecfe7d
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 154,\n         \"digest
\": \"sha256:67012cca8f31dc3b8ee2305e7762fee20c250513effdedb38a1c37784a5a2e71\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 176,\n         \"digest
\": \"sha256:3bc892145603fffc9b1c97c94e2985b4cb19ca508750b15845a5d97becbd1a0e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 183,\n         \"digest
\": \"sha256:6f1c79518f18251d35977e7e46bfa6c6b9cf50df2a79d4194941d95c54258d18\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest
\": \"sha256:b7bcfbc2e2888afebede4dd1cd5eebf029bb6315feeaf0b56e425e11a50afe42\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest\":
 \"sha256:2b220f8b0f32b7c2ed8eaafe1c802633bbd94849b9ab73926f0ba46cdae91629\"\n      }\n
   ]\n}"
 },
 "responseElements": {
  "image": {
   "repositoryName": "testrepo",
   "imageManifest": "{\n   \"schemaVersion\": 2,\n   \"mediaType\": \"application/
vnd.docker.distribution.manifest.v2+json\",\n   \"config\": {\n      \"mediaType\":
 \"application/vnd.docker.container.image.v1+json\",\n      \"size\": 5543,\n
 \"digest\": \"sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a
\"\n   },\n   \"layers\": [\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 43252507,\n
```

```
 \"digest\": \"sha256:3b37166ec61459e76e33282dda08f2a9cd698ca7e3d6bc44e6a6e7580cdeff8e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 846,\n         \"digest
\": \"sha256:504facff238fde83f1ca8f9f54520b4219c5b8f80be9616ddc52d31448a044bd
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 615,\n         \"digest
\": \"sha256:ebbcacd28e101968415b0c812b2d2dc60f969e36b0b08c073bf796e12b1bb449\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 850,\n         \"digest
\": \"sha256:c7fb3351ecad291a88b92b600037e2435c84a347683d540042086fe72c902b8a
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 168,\n         \"digest\":
 \"sha256:2e3debadcbf7e542e2aefbce1b64a358b1931fb403b3e4aeca27cb4d809d56c2\"\n      },
\n      {\n         \"mediaType\": \"application/vnd.docker.image.rootfs.diff.tar.gzip
\",\n         \"size\": 37720774,\n         \"digest\":
 \"sha256:f8c9f51ad524d8ae9bf4db69cd3e720ba92373ec265f5c390ffb21bb0c277941\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 30432107,\n
 \"digest\": \"sha256:813a50b13f61cf1f8d25f19fa96ad3aa5b552896c83e86ce413b48b091d7f01b
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 197,\n         \"digest
\": \"sha256:7ab043301a6187ea3293d80b30ba06c7bf1a0c3cd4c43d10353b31bc0cecfe7d
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 154,\n         \"digest
\": \"sha256:67012cca8f31dc3b8ee2305e7762fee20c250513effdedb38a1c37784a5a2e71\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 176,\n         \"digest
\": \"sha256:3bc892145603fffc9b1c97c94e2985b4cb19ca508750b15845a5d97becbd1a0e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 183,\n         \"digest
\": \"sha256:6f1c79518f18251d35977e7e46bfa6c6b9cf50df2a79d4194941d95c54258d18\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest
\": \"sha256:b7bcfbc2e2888afebede4dd1cd5eebf029bb6315feeaf0b56e425e11a50afe42\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest\":
 \"sha256:2b220f8b0f32b7c2ed8eaafe1c802633bbd94849b9ab73926f0ba46cdae91629\"\n      }\n
   ]\n}",
   "registryId": "123456789012",
   "imageId": {
    "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e",
    "imageTag": "latest"
   }
```

```
  }
 },
 "requestID": "cf044b7d-5f9d-11e9-9b2a-95983139cc57",
 "eventID": "2bfd4ee2-2178-4a82-a27d-b12939923f0f",
 "resources": [{
  "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
  "accountId": "123456789012"
 }],
 "eventType": "AwsApiCall",
 "recipientAccountId": "123456789012"
}
```

例: イメージプルアクション

以下の例は、BatchGetImage アクションを使用するイメージのプルの CloudTrail ログエントリを示しています。

> **Note**
>
> イメージをプルすると、すでにイメージがローカルにない場合、GetDownloadUrlForLayer の参照もCloudTrail ログに表示されます。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
    "arn": "arn:aws:sts::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T17:23:20Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "BatchGetImage",
```

```
 "awsRegion": "us-east-2",
 "sourceIPAddress": "ecr.amazonaws.com",
 "userAgent": "ecr.amazonaws.com",
 "requestParameters": {
  "imageIds": [{
   "imageTag": "latest"
  }],
  "acceptedMediaTypes": [
   "application/json",
   "application/vnd.oci.image.manifest.v1+json",
   "application/vnd.oci.image.index.v1+json",
   "application/vnd.docker.distribution.manifest.v2+json",
   "application/vnd.docker.distribution.manifest.list.v2+json",
   "application/vnd.docker.distribution.manifest.v1+prettyjws"
  ],
  "repositoryName": "testrepo",
  "registryId": "123456789012"
 },
 "responseElements": null,
 "requestID": "2a1b97ee-5fa3-11e9-a8cd-cd2391aeda93",
 "eventID": "c84f5880-c2f9-4585-9757-28fa5c1065df",
 "resources": [{
  "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
  "accountId": "123456789012"
 }],
 "eventType": "AwsApiCall",
 "recipientAccountId": "123456789012"
}
```


例: イメージライフサイクルポリシーアクション

次の例は、ライフサイクルポリシールールによってイメージの期限が切れたことを示す CloudTrail ログエントリを示しています。このイベントタイプは、イベント名フィールドの `PolicyExecutionEvent` をフィルタリングすることによって検索できます。

ライフサイクルポリシーのプレビューをテストすると、Amazon ECR は、`PolicyExecutionEvent` とまったく同じ構造で、`DryRunEvent` のイベント名フィールドが CloudTrail ログエントリを生成します。イベント名を `DryRunEvent` に変更することで、代わりにドライランイベントをフィルタリングできます。

> **Important**
>
> 個々の CloudTrail イベントのサイズ制限により、10 個以上のイメージが期限切れになるライフサイクルポリシーアクションの場合、Amazon ECR が複数のイベントを CloudTrail に送信します。また、Amazon ECR には、イメージごとに最大 100 個のタグが含まれます。


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "accountId": "123456789012",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-03-12T20:22:12Z",
    "eventSource": "ecr.amazonaws.com",
    "eventName": "PolicyExecutionEvent",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "9354dd7f-9aac-4e9d-956d-12561a4923aa",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:ecr:us-west-2:123456789012:repository/testrepo",
            "accountId": "123456789012",
            "type": "AWS::ECR::Repository"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "123456789012",
    "serviceEventDetails": {
        "repositoryName": "testrepo",
        "lifecycleEventPolicy": {
            "lifecycleEventRules": [
                {
                    "rulePriority": 1,
                    "description": "remove all images > 2",
                    "lifecycleEventSelection": {
                        "tagStatus": "Any",
                        "tagPrefixList": [],
```

```
                        "countType": "Image count more than",
                        "countNumber": 2
                    },
                    "action": "expire"
                }
            ],
            "lastEvaluatedAt": 0,
            "policyVersion": 1,
            "policyId": "ceb86829-58e7-9498-920c-aa042e33037b"
        },
        "lifecycleEventImageActions": [
            {
                "lifecycleEventImage": {
                    "digest":
"sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45",
                    "tagStatus": "Tagged",
                    "tagList": [
                        "alpine"
                    ],
                    "pushedAt": 1584042813000
                },
                "rulePriority": 1
            },
            {
                "lifecycleEventImage": {
                    "digest":
"sha256:6ab380c5a5acf71c1b6660d645d2cd79cc8ce91b38e0352cbf9561e050427baf",
                    "tagStatus": "Tagged",
                    "tagList": [
                        "centos"
                    ],
                    "pushedAt": 1584042842000
                },
                "rulePriority": 1
            }
        ],
        "lifecycleEventFailureDetails": [
            {
                "lifecycleEventImage": {
                    "digest":
"sha256:9117e1bc28cd20751e584b4ccd19b1178d14cf02d134b04ce6be0cc51bff762a",
                    "tagStatus": "Untagged",
                    "tagList": [],
                    "pushedAt": 1584042844000
```

```
                },
                "rulePriority": 1,
                "failureCode": "ImageReferencedByManifestList",
                "failureReason": "Requested image referenced by manifest list:
 [sha256:4b27c83d44a18c31543039d9e8b2786043ec6c8d00804d5800c5148d6b6f65bc]"
            }
        ]
    }
}
```

例: イメージアーカイブアクション

次の例は、 を targetStorageClassに設定して UpdateImageStorageClassアクションを使用してアーカイブされているイメージを示す CloudTrail ログエントリを示していますARCHIVE。

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T16:45:00Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "UpdateImageStorageClass",
 "awsRegion": "us-east-2",
 "sourceIPAddress": "AWS Internal",
 "userAgent": "AWS Internal",
 "requestParameters": {
  "repositoryName": "testrepo",
  "imageId": {
   "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e"
  },
```

```
  "targetStorageClass": "ARCHIVE",
  "registryId": "123456789012"
 },
 "responseElements": {
  "image": {
   "registryId": "123456789012",
   "repositoryName": "testrepo",
   "imageId": {
    "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e"
   },
   "imageStatus": "ARCHIVED"
  }
 },
 "requestID": "cf044b7d-EXAMPLE",
 "eventID": "2bfd4ee2-EXAMPLE",
 "readOnly": false,
 "resources": [{
  "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
  "accountId": "123456789012"
 }],
 "eventType": "AwsApiCall",
 "managementEvent": true,
 "recipientAccountId": "123456789012",
 "eventCategory": "Management"
}
```

例: イメージ復元アクション

次の例は、復元されるイメージを示す CloudTrail ログエントリを示しています。アーカイブされたイメージを復元すると、次の 2 つのイベントが生成されます。

1. 復元が開始されたときの API コールイベント
2. 非同期復元オペレーションが完了したときのサービスイベント

API コールイベント (復元開始)

次の例は、 を targetStorageClassに設定して UpdateImageStorageClassアクションを使用してイメージを復元するための最初の API コールを示していますSTANDARD。レスポンスには、イメージのステータスが と表示されますACTIVATING。

```
{
```

```
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T16:45:00Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "UpdateImageStorageClass",
 "awsRegion": "us-east-2",
 "sourceIPAddress": "AWS Internal",
 "userAgent": "AWS Internal",
 "requestParameters": {
  "repositoryName": "testrepo",
  "imageId": {
   "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e"
  },
  "targetStorageClass": "STANDARD",
  "registryId": "123456789012"
 },
 "responseElements": {
  "image": {
   "registryId": "123456789012",
   "repositoryName": "testrepo",
   "imageId": {
    "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e"
   },
   "imageStatus": "ACTIVATING"
  }
 },
 "requestID": "cf044b7d-EXAMPLE",
 "eventID": "2bfd4ee2-EXAMPLE",
 "readOnly": false,
```

```
 "resources": [{
  "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
  "accountId": "123456789012"
 }],
 "eventType": "AwsApiCall",
 "managementEvent": true,
 "recipientAccountId": "123456789012",
 "eventCategory": "Management"
}
```

サービスイベント (復元の完了)

次の例は、非同期復元オペレーションの完了時に生成されるサービスイベントを示しています。このイベントタイプは、イベント名フィールドの ImageActivationEvent をフィルタリングすることによって検索できます。serviceEventDetails セクションには、復元結果と最終的なイメージステータスが含まれます。

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "accountId": "123456789012",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-03-12T20:22:12Z",
    "eventSource": "ecr.amazonaws.com",
    "eventName": "ImageActivationEvent",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "9354dd7f-EXAMPLE",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:ecr:us-west-2:123456789012:repository/testrepo",
            "accountId": "123456789012",
            "type": "AWS::ECR::Repository"
        }
    ],
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
```

```
    "serviceEventDetails": {
        "repositoryName": "testrepo",
        "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e",
        "targetStorageClass": "STANDARD",
        "result": "SUCCESS",
        "imageStatus": "ACTIVE"
    },
    "eventCategory": "Management"
}
```

例: イメージリファラーアクション

次の例は、OCI 1.1準拠クライアントが Referrers API を使用してイメージのリファラーまたはリファレンスアーティファクトのリストを取得するタイミングを示す AWS CloudTrail ログエントリを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-10-08T16:38:39Z",
                "mfaAuthenticated": "false"
            },
            "ec2RoleDelivery": "2.0"
        },
        "invokedBy": "ecr.amazonaws.com"
    },
    "eventTime": "2024-10-08T17:22:51Z",
```

```
    "eventSource": "ecr.amazonaws.com",
    "eventName": "ListImageReferrers",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "ecr.amazonaws.com",
    "userAgent": "ecr.amazonaws.com",
    "requestParameters": {
        "registryId": "123456789012",
        "repositoryName": "testrepo",
        "subjectId": {
            "imageDigest":
 "sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a"
        },
        "nextToken": "urD72mdD/mC8b5-EXAMPLE"
    },
    "responseElements": null,
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "123456789012",
            "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```

AWS SDK での Amazon ECR 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

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。

AWS SDKsコード例

次のコード例は、 AWS Software Development Kit (SDK) で Amazon ECR を使用する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

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を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。

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 Amazon ECR 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

- AWS SDKs基本的な例
  - Hello Amazon ECR
  - AWS SDK を使用した Amazon ECR の基本について説明します。
  - AWS SDKsアクション
    - AWS SDK または CLI CreateRepositoryで を使用する
    - AWS SDK または CLI DeleteRepositoryで を使用する
    - AWS SDK または CLI DescribeImagesで を使用する
    - AWS SDK または CLI DescribeRepositoriesで を使用する
    - AWS SDK または CLI GetAuthorizationTokenで を使用する
    - AWS SDK または CLI GetRepositoryPolicyで を使用する
    - AWS SDK または CLI ListImagesで を使用する
    - AWS SDK PushImageCmdで を使用する
    - AWS SDK または CLI PutLifeCyclePolicyで を使用する
    - AWS SDK または CLI SetRepositoryPolicyで を使用する
    - AWS SDK または CLI StartLifecyclePolicyPreviewで を使用する

AWS SDKs基本的な例

次のコード例は、 AWS SDK での Amazon Elastic Container Registry の基本的な使用方法を示しています。

例

- Hello Amazon ECR
- AWS SDK を使用した Amazon ECR の基本について説明します。
- AWS SDKsアクション
  - AWS SDK または CLI CreateRepositoryで を使用する
  - AWS SDK または CLI DeleteRepositoryで を使用する
  - AWS SDK または CLI DescribeImagesで を使用する
  - AWS SDK または CLI DescribeRepositoriesで を使用する
  - AWS SDK または CLI GetAuthorizationTokenで を使用する
  - AWS SDK または CLI GetRepositoryPolicyで を使用する
  - AWS SDK または CLI ListImagesで を使用する
  - AWS SDK PushImageCmdで を使用する
  - AWS SDK または CLI PutLifeCyclePolicyで を使用する
  - AWS SDK または CLI SetRepositoryPolicyで を使用する
  - AWS SDK または CLI StartLifecyclePolicyPreviewで を使用する

Hello Amazon ECR

次のコード例は、Amazon ECR の使用を開始する方法を示しています。

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.ecr.EcrClient;
import software.amazon.awssdk.services.ecr.model.EcrException;
import software.amazon.awssdk.services.ecr.model.ListImagesRequest;
import software.amazon.awssdk.services.ecr.paginators.ListImagesIterable;

public class HelloECR {

    public static void main(String[] args) {
        final String usage = """
            Usage:    <repositoryName>

            Where:
               repositoryName - The name of the Amazon ECR repository.
            """;

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

        String repoName = args[0];
        EcrClient ecrClient = EcrClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listImageTags(ecrClient, repoName);
    }
    public static void listImageTags(EcrClient ecrClient, String repoName){
        ListImagesRequest listImagesPaginator = ListImagesRequest.builder()
            .repositoryName(repoName)
            .build();

        ListImagesIterable imagesIterable =
 ecrClient.listImagesPaginator(listImagesPaginator);
        imagesIterable.stream()
            .flatMap(r -> r.imageIds().stream())
            .forEach(image -> System.out.println("The docker image tag is: "
 +image.imageTag()));
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「listImages」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。

```
import aws.sdk.kotlin.services.ecr.EcrClient
import aws.sdk.kotlin.services.ecr.model.ListImagesRequest
import kotlin.system.exitProcess

suspend fun main(args: Array<String>) {
    val usage = """
            Usage: <repositoryName>

            Where:
               repositoryName - The name of the Amazon ECR repository.

    """.trimIndent()

    if (args.size != 1) {
        println(usage)
        exitProcess(1)
    }

    val repoName = args[0]
    listImageTags(repoName)
}

suspend fun listImageTags(repoName: String?) {
    val listImages =
        ListImagesRequest {
            repositoryName = repoName
        }
```

```
    EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
        val imageResponse = ecrClient.listImages(listImages)
        imageResponse.imageIds?.forEach { imageId ->
            println("Image tag: ${imageId.imageTag}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「listImages」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。

```
import boto3
import argparse
from boto3 import client


def hello_ecr(ecr_client: client, repository_name: str) -> None:
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon Elastic Container
 Registry (Amazon ECR)
    client and list the images in a repository.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param ecr_client: A Boto3 Amazon ECR Client object. This object wraps
                           the low-level Amazon ECR service API.
    :param repository_name: The name of an Amazon ECR repository in your account.
    """
    print(
```

```
        f"Hello, Amazon ECR! Let's list some images in the repository
 '{repository_name}':\n"
    )
    paginator = ecr_client.get_paginator("list_images")
    page_iterator = paginator.paginate(
        repositoryName=repository_name, PaginationConfig={"MaxItems": 10}
    )

    image_names: [str] = []
    for page in page_iterator:
        for schedule in page["imageIds"]:
            image_names.append(schedule["imageTag"])

    print(f"{len(image_names)} image(s) retrieved.")
    for schedule_name in image_names:
        print(f"\t{schedule_name}")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Run hello Amazon ECR.")
    parser.add_argument(
        "--repository-name",
        type=str,
        help="the name of an Amazon ECR repository in your account.",
        required=True,
    )
    args = parser.parse_args()

    hello_ecr(boto3.client("ecr"), args.repository_name)
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「listImages」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 Amazon ECR 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 を使用した Amazon ECR の基本について説明します。

次のコード例は、以下を実行する方法を示しています。

- Amazon ECR リポジトリを作成します。
- リポジトリポリシーを設定します。
- リポジトリ URI を取得します。
- Amazon ECR 認可トークンを取得します。
- Amazon ECR リポジトリのライフサイクルポリシーを設定します。
- Docker イメージを Amazon ECR リポジトリにプッシュします。
- Amazon ECR リポジトリにイメージが存在しているか確認します。
- アカウントの Amazon ECR リポジトリをリストし、その詳細を取得します。
- Amazon ECR リポジトリを削除します。

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。

Amazon ECR 機能を示すインタラクティブなシナリオを実行します。

```
import software.amazon.awssdk.services.ecr.model.EcrException;
import
 software.amazon.awssdk.services.ecr.model.RepositoryPolicyNotFoundException;

import java.util.Scanner;

/**
 * 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
 *
```

```
 * This Java code example requires an IAM Role that has permissions to interact
 with the Amazon ECR service.
 *
 * To create an IAM role, see:
 *
 * https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create.html
 *
 * This Java scenario example requires a local docker image named echo-text.
 Without a local image,
 * this Java program will not successfully run. For more information including
 how to create the local
 * image, see:
 *
 * /scenarios/basics/ecr/README
 *
 */
public class ECRScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");
    public static void main(String[] args) {
        final String usage = """
            Usage: <iamRoleARN> <accountId>

            Where:
               iamRoleARN - The IAM role ARN that has the necessary permissions
 to access and manage the Amazon ECR repository.
               accountId - Your AWS account number.
            """;

        if (args.length != 2) {
            System.out.println(usage);
            return;
        }

        ECRActions ecrActions = new ECRActions();
        String iamRole = args[0];
        String accountId = args[1];
        String localImageName;

        Scanner scanner = new Scanner(System.in);
        System.out.println("""
             The Amazon Elastic Container Registry (ECR) is a fully-managed
 Docker container registry
```

```
            service provided by AWS. It allows developers and organizations to
securely
            store, manage, and deploy Docker container images.
            ECR provides a simple and scalable way to manage container images
throughout their lifecycle,
            from building and testing to production deployment.\s

            The `EcrAsyncClient` interface in the AWS SDK for Java 2.x provides
a set of methods to
            programmatically interact with the Amazon ECR service. This allows
developers to
            automate the storage, retrieval, and management of container images
as part of their application
            deployment pipelines. With ECR, teams can focus on building and
deploying their
            applications without having to worry about the underlying
infrastructure required to
            host and manage a container registry.

           This scenario walks you through how to perform key operations for
this service.
           Let's get started...

           You have two choices:
           1 - Run the entire program.
           2 - Delete an existing Amazon ECR repository named echo-text (created
from a previous execution of
           this program that did not complete).
           """);

       while (true) {
           String input = scanner.nextLine();
           if (input.trim().equalsIgnoreCase("1")) {
               System.out.println("Continuing with the program...");
               System.out.println("");
               break;
           } else if (input.trim().equalsIgnoreCase("2")) {
               String repoName = "echo-text";
               ecrActions.deleteECRRepository(repoName);
               return;
           } else {
               // Handle invalid input.
               System.out.println("Invalid input. Please try again.");
           }
```

```
        }

        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println("""
           1. Create an ECR repository.

           The first task is to ensure we have a local Docker image named echo-
text.
           If this image exists, then an Amazon ECR repository is created.

           An ECR repository is a private Docker container repository provided
           by Amazon Web Services (AWS). It is a managed service that makes it
 easy
           to store, manage, and deploy Docker container images.\s
           """ );

        // Ensure that a local docker image named echo-text exists.
        boolean doesExist = ecrActions.isEchoTextImagePresent();
        String repoName;
        if (!doesExist){
            System.out.println("The local image named echo-text does not exist");
            return;
        } else {
            localImageName = "echo-text";
            repoName = "echo-text";
        }

        try {
            String repoArn = ecrActions.createECRRepository(repoName);
            System.out.println("The ARN of the ECR repository is " + repoArn);

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid repository name: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred while creating the ECR
 repository: " + e.getMessage());
            e.printStackTrace();
            return;
        }
        waitForInputToContinue(scanner);
```

```
        System.out.println(DASHES);
        System.out.println("""
        2. Set an ECR repository policy.

        Setting an ECR repository policy using the `setRepositoryPolicy` function
is crucial for maintaining
        the security and integrity of your container images. The repository
policy allows you to
        define specific rules and restrictions for accessing and managing the
images stored within your ECR
        repository.
        """);
        waitForInputToContinue(scanner);
        try {
            ecrActions.setRepoPolicy(repoName, iamRole);

        } catch (RepositoryPolicyNotFoundException e) {
            System.err.println("Invalid repository name: " + e.getMessage());
            return;
        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred while creating the ECR
repository: " + e.getMessage());
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        3. Display ECR repository policy.

        Now we will retrieve the ECR policy to ensure it was successfully set.
        """);
        waitForInputToContinue(scanner);
        try {
            String policyText = ecrActions.getRepoPolicy(repoName);
            System.out.println("Policy Text:");
            System.out.println(policyText);

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
```

```
        } catch (RuntimeException e) {
            System.err.println("An error occurred while creating the ECR
repository: " + e.getMessage());
            return;
        }

        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        4. Retrieve an ECR authorization token.

        You need an authorization token to securely access and interact with the
Amazon ECR registry.
        The `getAuthorizationToken` method of the `EcrAsyncClient` is responsible
for securely accessing
        and interacting with an Amazon ECR repository. This operation is
responsible for obtaining a
        valid authorization token, which is required to authenticate your
requests to the ECR service.

        Without a valid authorization token, you would not be able to perform any
operations on the
        ECR repository, such as pushing, pulling, or managing your Docker images.

        """);
        waitForInputToContinue(scanner);
        try {
             ecrActions.getAuthToken();

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred while retrieving the
authorization token: " + e.getMessage());
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        5. Get the ECR Repository URI.
```

```
        The URI  of an Amazon ECR repository is important. When you want to
deploy a container image to
        a container orchestration platform like Amazon Elastic Kubernetes Service
(EKS)
        or Amazon Elastic Container Service (ECS), you need to specify the full
image URI,
        which includes the ECR repository URI. This allows the container runtime
to pull the
        correct container image from the ECR repository.
       """);
        waitForInputToContinue(scanner);

        try {
            ecrActions.getRepositoryURI(repoName);

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;

        } catch (RuntimeException e) {
            System.err.println("An error occurred while retrieving the URI: " +
e.getMessage());
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
            6. Set an ECR Lifecycle Policy.

            An ECR Lifecycle Policy is used to manage the lifecycle of Docker
images stored in your ECR repositories.
            These policies allow you to automatically remove old or unused Docker
images from your repositories,
            freeing up storage space and reducing costs.

            This example policy helps to maintain the size and efficiency of the
container registry
            by automatically removing older and potentially unused images,
ensuring that the
            storage is optimized and the registry remains up-to-date.
            """);
        waitForInputToContinue(scanner);
        try {
```

```
            ecrActions.setLifeCyclePolicy(repoName);

        } catch (RuntimeException e) {
            System.err.println("An error occurred while setting the lifecycle
policy: " + e.getMessage());
            e.printStackTrace();
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        7. Push a docker image to the Amazon ECR Repository.

        The `pushImageCmd()` method pushes a local Docker image to an Amazon ECR
repository.
        It sets up the Docker client by connecting to the local Docker host using
the default port.
        It then retrieves the authorization token for the ECR repository by
making a call to the AWS SDK.

        The method uses the authorization token to create an `AuthConfig` object,
which is used to authenticate
        the Docker client when pushing the image. Finally, the method tags the
Docker image with the specified
        repository name and image tag, and then pushes the image to the ECR
repository using the Docker client.
        If the push operation is successful, the method prints a message
indicating that the image was pushed to ECR.
        """);
        waitForInputToContinue(scanner);

        try {
            ecrActions.pushDockerImage(repoName, localImageName);

        } catch (RuntimeException e) {
            System.err.println("An error occurred while pushing a local Docker
image to Amazon ECR: " + e.getMessage());
            e.printStackTrace();
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
```

```
        System.out.println("8. Verify if the image is in the ECR Repository.");
        waitForInputToContinue(scanner);
        try {
            ecrActions.verifyImage(repoName, localImageName);

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred " + e.getMessage());
            e.printStackTrace();
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("9. As an optional step, you can interact with the
 image in Amazon ECR by using the CLI.");
        System.out.println("Would you like to view instructions on how to use the
 CLI to run the image? (y/n)");
        String ans = scanner.nextLine().trim();
        if (ans.equalsIgnoreCase("y")) {
            String instructions = """
            1. Authenticate with ECR - Before you can pull the image from Amazon
 ECR, you need to authenticate with the registry. You can do this using the AWS
 CLI:

                aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin %s.dkr.ecr.us-east-1.amazonaws.com

            2. Describe the image using this command:

               aws ecr describe-images --repository-name %s --image-ids imageTag=
%s

            3. Run the Docker container and view the output using this command:

               docker run --rm %s.dkr.ecr.us-east-1.amazonaws.com/%s:%s
            """;

            instructions = String.format(instructions, accountId, repoName,
 localImageName, accountId, repoName, localImageName);
            System.out.println(instructions);
        }
```

```
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("10. Delete the ECR Repository.");
        System.out.println(
        """
        If the repository isn't empty, you must either delete the contents of the
repository
        or use the force option (used in this scenario) to delete the repository
and have Amazon ECR delete all of its contents
        on your behalf.
        """);
        System.out.println("Would you like to delete the Amazon ECR Repository?
(y/n)");
        String delAns = scanner.nextLine().trim();
        if (delAns.equalsIgnoreCase("y")) {
            System.out.println("You selected to delete the AWS ECR resources.");

            try {
                ecrActions.deleteECRRepository(repoName);

            } catch (EcrException e) {
                System.err.println("An ECR exception occurred: " +
e.getMessage());
                return;
            } catch (RuntimeException e) {
                System.err.println("An error occurred while deleting the Docker
image: " + e.getMessage());
                e.printStackTrace();
                return;
            }
        }

        System.out.println(DASHES);
        System.out.println("This concludes the Amazon ECR SDK scenario");
        System.out.println(DASHES);
    }

  private static void waitForInputToContinue(Scanner scanner) {
      while (true) {
          System.out.println("");
          System.out.println("Enter 'c' followed by <ENTER> to continue:");
          String input = scanner.nextLine();
```

```
            if (input.trim().equalsIgnoreCase("c")) {
                System.out.println("Continuing with the program...");
                System.out.println("");
                break;
            } else {
                // Handle invalid input.
                System.out.println("Invalid input. Please try again.");
            }
        }
    }
}
```

Amazon ECR SDK メソッドのラッパークラス。

```
import com.github.dockerjava.api.DockerClient;
import com.github.dockerjava.api.exception.DockerClientException;
import com.github.dockerjava.api.model.AuthConfig;
import com.github.dockerjava.api.model.Image;
import com.github.dockerjava.core.DockerClientBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.http.async.SdkAsyncHttpClient;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ecr.EcrAsyncClient;
import software.amazon.awssdk.services.ecr.model.AuthorizationData;
import software.amazon.awssdk.services.ecr.model.CreateRepositoryRequest;
import software.amazon.awssdk.services.ecr.model.CreateRepositoryResponse;
import software.amazon.awssdk.services.ecr.model.DeleteRepositoryRequest;
import software.amazon.awssdk.services.ecr.model.DeleteRepositoryResponse;
import software.amazon.awssdk.services.ecr.model.DescribeImagesRequest;
import software.amazon.awssdk.services.ecr.model.DescribeImagesResponse;
import software.amazon.awssdk.services.ecr.model.DescribeRepositoriesRequest;
import software.amazon.awssdk.services.ecr.model.DescribeRepositoriesResponse;
import software.amazon.awssdk.services.ecr.model.EcrException;
import software.amazon.awssdk.services.ecr.model.GetAuthorizationTokenResponse;
import software.amazon.awssdk.services.ecr.model.GetRepositoryPolicyRequest;
import software.amazon.awssdk.services.ecr.model.GetRepositoryPolicyResponse;
import software.amazon.awssdk.services.ecr.model.ImageIdentifier;
import software.amazon.awssdk.services.ecr.model.Repository;
```

```
import
 software.amazon.awssdk.services.ecr.model.RepositoryPolicyNotFoundException;
import software.amazon.awssdk.services.ecr.model.SetRepositoryPolicyRequest;
import software.amazon.awssdk.services.ecr.model.SetRepositoryPolicyResponse;
import
 software.amazon.awssdk.services.ecr.model.StartLifecyclePolicyPreviewRequest;
import
 software.amazon.awssdk.services.ecr.model.StartLifecyclePolicyPreviewResponse;
import com.github.dockerjava.api.command.DockerCmdExecFactory;
import com.github.dockerjava.netty.NettyDockerCmdExecFactory;
import java.time.Duration;
import java.util.Base64;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionException;

public class ECRActions {
    private static EcrAsyncClient ecrClient;

    private static DockerClient dockerClient;

    private static Logger logger = LoggerFactory.getLogger(ECRActions.class);

    /**
     * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
     *
     * @param repoName the name of the repository to create.
     * @return the Amazon Resource Name (ARN) of the created repository, or an
 empty string if the operation failed.
     * @throws IllegalArgumentException     If repository name is invalid.
     * @throws RuntimeException             if an error occurs while creating the
 repository.
     */
    public String createECRRepository(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
 empty");
        }

        CreateRepositoryRequest request = CreateRepositoryRequest.builder()
            .repositoryName(repoName)
            .build();
```

```
        CompletableFuture<CreateRepositoryResponse> response =
getAsyncClient().createRepository(request);
        try {
            CreateRepositoryResponse result = response.join();
            if (result != null) {
                System.out.println("The " + repoName + " repository was created
successfully.");
                return result.repository().repositoryArn();
            } else {
                throw new RuntimeException("Unexpected response type");
            }
        } catch (CompletionException e) {
            Throwable cause = e.getCause();
            if (cause instanceof EcrException ex) {
                if
("RepositoryAlreadyExistsException".equals(ex.awsErrorDetails().errorCode())) {
                    System.out.println("The Amazon ECR repository already exists,
moving on...");
                    DescribeRepositoriesRequest describeRequest =
DescribeRepositoriesRequest.builder()
                        .repositoryNames(repoName)
                        .build();
                    DescribeRepositoriesResponse describeResponse =
getAsyncClient().describeRepositories(describeRequest).join();
                    return
describeResponse.repositories().get(0).repositoryArn();
                } else {
                    throw new RuntimeException(ex);
                }
            } else {
                throw new RuntimeException(e);
            }
        }
    }

    /**
     * Deletes an ECR (Elastic Container Registry) repository.
     *
     * @param repoName the name of the repository to delete.
     * @throws IllegalArgumentException if the repository name is null or empty.
     * @throws EcrException if there is an error deleting the repository.
     * @throws RuntimeException if an unexpected error occurs during the deletion
process.
     */
```

```
   public void deleteECRRepository(String repoName) {
       if (repoName == null || repoName.isEmpty()) {
           throw new IllegalArgumentException("Repository name cannot be null or
empty");
       }

       DeleteRepositoryRequest repositoryRequest =
DeleteRepositoryRequest.builder()
           .force(true)
           .repositoryName(repoName)
           .build();

       CompletableFuture<DeleteRepositoryResponse> response =
getAsyncClient().deleteRepository(repositoryRequest);
       response.whenComplete((deleteRepositoryResponse, ex) -> {
           if (deleteRepositoryResponse != null) {
               System.out.println("You have successfully deleted the " +
repoName + " repository");
           } else {
               Throwable cause = ex.getCause();
               if (cause instanceof EcrException) {
                   throw (EcrException) cause;
               } else {
                   throw new RuntimeException("Unexpected error: " +
cause.getMessage(), cause);
               }
           }
       });

       // Wait for the CompletableFuture to complete
       response.join();
   }



   private static DockerClient getDockerClient() {
       String osName = System.getProperty("os.name");
       if (osName.startsWith("Windows")) {
           // Make sure Docker Desktop is running.
           String dockerHost = "tcp://localhost:2375"; // Use the Docker Desktop
default port.
           DockerCmdExecFactory dockerCmdExecFactory = new
NettyDockerCmdExecFactory().withReadTimeout(20000).withConnectTimeout(20000);
```

```
            dockerClient =
DockerClientBuilder.getInstance(dockerHost).withDockerCmdExecFactory(dockerCmdExecFactor
        } else {
            dockerClient = DockerClientBuilder.getInstance().build();
        }
        return dockerClient;
    }

    /**
     * Retrieves an asynchronous Amazon Elastic Container Registry (ECR) client.
     *
     * @return the configured ECR asynchronous client.
     */
    private static EcrAsyncClient getAsyncClient() {

        /*
         The `NettyNioAsyncHttpClient` class is part of the AWS SDK for Java,
version 2,
         and it is designed to provide a high-performance, asynchronous HTTP
client for interacting with AWS services.
         It uses the Netty framework to handle the underlying network
communication and the Java NIO API to
         provide a non-blocking, event-driven approach to HTTP requests and
responses.
         */
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(50)  // Adjust as needed.
            .connectionTimeout(Duration.ofSeconds(60))  // Set the connection
timeout.
            .readTimeout(Duration.ofSeconds(60))  // Set the read timeout.
            .writeTimeout(Duration.ofSeconds(60))  // Set the write timeout.
            .build();

        ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))  // Set the overall API call
timeout.
            .apiCallAttemptTimeout(Duration.ofSeconds(90))  // Set the individual
call attempt timeout.
            .build();

        if (ecrClient == null) {
            ecrClient = EcrAsyncClient.builder()
                .region(Region.US_EAST_1)
```

```
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return ecrClient;
    }

    /**
     * Sets the lifecycle policy for the specified repository.
     *
     * @param repoName the name of the repository for which to set the lifecycle
 policy.
     */
    public void setLifeCyclePolicy(String repoName) {
        /*
           This policy helps to maintain the size and efficiency of the container
 registry
           by automatically removing older and potentially unused images,
           ensuring that the storage is optimized and the registry remains up-to-
date.
         */
        String polText = """
             {
             "rules": [
                 {
                     "rulePriority": 1,
                     "description": "Expire images older than 14 days",
                     "selection": {
                         "tagStatus": "any",
                         "countType": "sinceImagePushed",
                         "countUnit": "days",
                         "countNumber": 14
                     },
                     "action": {
                         "type": "expire"
                     }
                 }
            ]
            }
            """;

        StartLifecyclePolicyPreviewRequest lifecyclePolicyPreviewRequest =
 StartLifecyclePolicyPreviewRequest.builder()
            .lifecyclePolicyText(polText)
```

```
            .repositoryName(repoName)
            .build();

        CompletableFuture<StartLifecyclePolicyPreviewResponse> response =
getAsyncClient().startLifecyclePolicyPreview(lifecyclePolicyPreviewRequest);
        response.whenComplete((lifecyclePolicyPreviewResponse, ex) -> {
            if (lifecyclePolicyPreviewResponse != null) {
                System.out.println("Lifecycle policy preview started
successfully.");
            } else {
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
ex.getMessage();
                    throw new RuntimeException(errorMessage, ex);
                }
            }
        });
        // Wait for the CompletableFuture to complete.
        response.join();
    }

    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
(Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     * @throws EcrException             if there is an error retrieving the image
information from Amazon ECR.
     * @throws CompletionException      if the asynchronous operation completes
exceptionally.
     */
    public void verifyImage(String repositoryName, String imageTag) {
        DescribeImagesRequest request = DescribeImagesRequest.builder()
            .repositoryName(repositoryName)
            .imageIds(ImageIdentifier.builder().imageTag(imageTag).build())
            .build();

        CompletableFuture<DescribeImagesResponse> response =
getAsyncClient().describeImages(request);
        response.whenComplete((describeImagesResponse, ex) -> {
            if (ex != null) {
```

```
                if (ex instanceof CompletionException) {
                    Throwable cause = ex.getCause();
                    if (cause instanceof EcrException) {
                        throw (EcrException) cause;
                    } else {
                        throw new RuntimeException("Unexpected error: " +
 cause.getMessage(), cause);
                    }
                } else {
                    throw new RuntimeException("Unexpected error: " +
 ex.getCause());
                }
            } else if (describeImagesResponse != null && !
describeImagesResponse.imageDetails().isEmpty()) {
                System.out.println("Image is present in the repository.");
            } else {
                System.out.println("Image is not present in the repository.");
            }
        });

        // Wait for the CompletableFuture to complete.
        response.join();
    }

    /**
     * Retrieves the repository URI for the specified repository name.
     *
     * @param repoName the name of the repository to retrieve the URI for.
     * @return the repository URI for the specified repository name.
     * @throws EcrException        if there is an error retrieving the repository
 information.
     * @throws CompletionException if the asynchronous operation completes
 exceptionally.
     */
    public void getRepositoryURI(String repoName) {
        DescribeRepositoriesRequest request =
 DescribeRepositoriesRequest.builder()
            .repositoryNames(repoName)
            .build();

        CompletableFuture<DescribeRepositoriesResponse> response =
 getAsyncClient().describeRepositories(request);
        response.whenComplete((describeRepositoriesResponse, ex) -> {
            if (ex != null) {
```

```
                Throwable cause = ex.getCause();
                if (cause instanceof InterruptedException) {
                    Thread.currentThread().interrupt();
                    String errorMessage = "Thread interrupted while waiting for
asynchronous operation: " + cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                } else if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
                    String errorMessage = "Unexpected error: " +
cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                }
            } else {
                if (describeRepositoriesResponse != null) {
                    if (!describeRepositoriesResponse.repositories().isEmpty()) {
                        String repositoryUri =
describeRepositoriesResponse.repositories().get(0).repositoryUri();
                        System.out.println("Repository URI found: " +
repositoryUri);
                    } else {
                        System.out.println("No repositories found for the given
name.");
                    }
                } else {
                    System.err.println("No response received from
describeRepositories.");
                }
            }
        });
        response.join();
    }

    /**
     * Retrieves the authorization token for Amazon Elastic Container Registry
(ECR).
     * This method makes an asynchronous call to the ECR client to retrieve the
authorization token.
     * If the operation is successful, the method prints the token to the
console.
     * If an exception occurs, the method handles the exception and prints the
error message.
     *
```

```
    * @throws EcrException     if there is an error retrieving the authorization
token from ECR.
    * @throws RuntimeException if there is an unexpected error during the
operation.
    */
   public void getAuthToken() {
       CompletableFuture<GetAuthorizationTokenResponse> response =
getAsyncClient().getAuthorizationToken();
       response.whenComplete((authorizationTokenResponse, ex) -> {
           if (authorizationTokenResponse != null) {
               AuthorizationData authorizationData =
authorizationTokenResponse.authorizationData().get(0);
               String token = authorizationData.authorizationToken();
               if (!token.isEmpty()) {
                   System.out.println("The token was successfully retrieved.");
               }
           } else {
               if (ex.getCause() instanceof EcrException) {
                   throw (EcrException) ex.getCause();
               } else {
                   String errorMessage = "Unexpected error occurred: " +
ex.getMessage();
                   throw new RuntimeException(errorMessage, ex); // Rethrow the
exception
               }
           }
       });
       response.join();
   }

   /**
    * Gets the repository policy for the specified repository.
    *
    * @param repoName the name of the repository.
    * @throws EcrException if an AWS error occurs while getting the repository
policy.
    */
   public String getRepoPolicy(String repoName) {
       if (repoName == null || repoName.isEmpty()) {
           throw new IllegalArgumentException("Repository name cannot be null or
empty");
       }
```

```
        GetRepositoryPolicyRequest getRepositoryPolicyRequest =
GetRepositoryPolicyRequest.builder()
            .repositoryName(repoName)
            .build();

        CompletableFuture<GetRepositoryPolicyResponse> response =
getAsyncClient().getRepositoryPolicy(getRepositoryPolicyRequest);
        response.whenComplete((resp, ex) -> {
            if (resp != null) {
                System.out.println("Repository policy retrieved successfully.");
            } else {
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
ex.getMessage();
                    throw new RuntimeException(errorMessage, ex);
                }
            }
        });

        GetRepositoryPolicyResponse result = response.join();
        return result != null ? result.policyText() : null;
    }

    /**
     * Sets the repository policy for the specified ECR repository.
     *
     * @param repoName the name of the ECR repository.
     * @param iamRole  the IAM role to be granted access to the repository.
     * @throws RepositoryPolicyNotFoundException if the repository policy does
not exist.
     * @throws EcrException                      if there is an unexpected error
setting the repository policy.
     */
    public void setRepoPolicy(String repoName, String iamRole) {
        /*
          This example policy document grants the specified AWS principal the
permission to perform the
          `ecr:BatchGetImage` action. This policy is designed to allow the
specified principal
          to retrieve Docker images from the ECR repository.
         */
        String policyDocumentTemplate = """
```

```
            {
             "Version":"2012-10-17",
             "Statement" : [ {
               "Sid" : "new statement",
               "Effect" : "Allow",
               "Principal" : {
                 "AWS" : "%s"
               },
               "Action" : "ecr:BatchGetImage"
             } ]
           }
            """;

       String policyDocument = String.format(policyDocumentTemplate, iamRole);
       SetRepositoryPolicyRequest setRepositoryPolicyRequest =
SetRepositoryPolicyRequest.builder()
           .repositoryName(repoName)
           .policyText(policyDocument)
           .build();

       CompletableFuture<SetRepositoryPolicyResponse> response =
getAsyncClient().setRepositoryPolicy(setRepositoryPolicyRequest);
       response.whenComplete((resp, ex) -> {
           if (resp != null) {
               System.out.println("Repository policy set successfully.");
           } else {
               Throwable cause = ex.getCause();
               if (cause instanceof RepositoryPolicyNotFoundException) {
                   throw (RepositoryPolicyNotFoundException) cause;
               } else if (cause instanceof EcrException) {
                   throw (EcrException) cause;
               } else {
                   String errorMessage = "Unexpected error: " +
cause.getMessage();
                   throw new RuntimeException(errorMessage, cause);
               }
           }
       });
       response.join();
   }

   /**
    * Pushes a Docker image to an Amazon Elastic Container Registry (ECR)
repository.
```

```
     *
     * @param repoName  the name of the ECR repository to push the image to.
     * @param imageName the name of the Docker image.
     */
    public void pushDockerImage(String repoName, String imageName) {
        System.out.println("Pushing " + imageName + " to Amazon ECR will take a
few seconds.");
        CompletableFuture<AuthConfig> authResponseFuture =
getAsyncClient().getAuthorizationToken()
            .thenApply(response -> {
                String token =
response.authorizationData().get(0).authorizationToken();
                String decodedToken = new
String(Base64.getDecoder().decode(token));
                String password = decodedToken.substring(4);

                DescribeRepositoriesResponse descrRepoResponse =
getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
descrRepoResponse.repositories().stream().filter(r ->
r.repositoryName().equals(repoName)).findFirst().orElse(null);
                assert repoData != null;
                String registryURL = repoData.repositoryUri().split("/")[0];

                AuthConfig authConfig = new AuthConfig()
                    .withUsername("AWS")
                    .withPassword(password)
                    .withRegistryAddress(registryURL);
                return authConfig;
            })
            .thenCompose(authConfig -> {
                DescribeRepositoriesResponse descrRepoResponse =
getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
descrRepoResponse.repositories().stream().filter(r ->
r.repositoryName().equals(repoName)).findFirst().orElse(null);
                getDockerClient().tagImageCmd(imageName + ":latest",
repoData.repositoryUri() + ":latest", imageName).exec();
                try {

getDockerClient().pushImageCmd(repoData.repositoryUri()).withTag("echo-
text").withAuthConfig(authConfig).start().awaitCompletion();
                    System.out.println("The " + imageName + " was pushed to
ECR");
```

```
                } catch (InterruptedException e) {
                    throw (RuntimeException) e.getCause();
                }
                return CompletableFuture.completedFuture(authConfig);
            });

        authResponseFuture.join();
    }

    // Make sure local image echo-text exists.
    public boolean isEchoTextImagePresent() {
        try {
            List<Image> images = getDockerClient().listImagesCmd().exec();
            boolean helloWorldFound = false;
            for (Image image : images) {
                String[] repoTags = image.getRepoTags();
                if (repoTags != null) {
                    for (String tag : repoTags) {
                        if (tag.startsWith("echo-text")) {
                            System.out.println(tag);
                            helloWorldFound = true;
                        }
                    }
                }
            }
            if (helloWorldFound) {
                System.out.println("The local image named echo-text exists.");
                return true;
            } else {
                System.out.println("The local image named echo-text does not
 exist.");
                return false;
            }
        } catch (DockerClientException ex) {
            logger.error("ERROR: " + ex.getMessage());
            return false;
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
  - CreateRepository
  - DeleteRepository
  - DescribeImages
  - DescribeRepositories
  - GetAuthorizationToken
  - GetRepositoryPolicy
  - SetRepositoryPolicy
  - StartLifecyclePolicyPreview

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。

Amazon ECR 機能を示すインタラクティブなシナリオを実行します。

```
import java.util.Scanner

/**
 * Before running this Kotlin 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-kotlin/latest/developer-guide/setup.html
 *
 * This code example requires an IAM Role that has permissions to interact with
 the Amazon ECR service.
 *
 * To create an IAM role, see:
 *
```

```
 * https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create.html
 *
 * This code example requires a local docker image named echo-text. Without a
 local image,
 * this program will not successfully run. For more information including how to
 create the local
 * image, see:
 *
 * /scenarios/basics/ecr/README
 *
 */

val DASHES = String(CharArray(80)).replace("\u0000", "-")

suspend fun main(args: Array<String>) {
    val usage =
        """
        Usage: <iamRoleARN> <accountId>

        Where:
           iamRoleARN - The IAM role ARN that has the necessary permissions to
 access and manage the Amazon ECR repository.
           accountId - Your AWS account number.

        """.trimIndent()

    if (args.size != 2) {
        println(usage)
        return
    }

    var iamRole = args[0]
    var localImageName: String
    var accountId = args[1]
    val ecrActions = ECRActions()
    val scanner = Scanner(System.`in`)

    println(
        """
        The Amazon Elastic Container Registry (ECR) is a fully-managed Docker
 container registry
        service provided by AWS. It allows developers and organizations to
 securely
        store, manage, and deploy Docker container images.
```

```
        ECR provides a simple and scalable way to manage container images
throughout their lifecycle,
        from building and testing to production deployment.

        The `EcrClient` service client that is part of the AWS SDK for Kotlin
provides a set of methods to
        programmatically interact with the Amazon ECR service. This allows
developers to
        automate the storage, retrieval, and management of container images as
part of their application
        deployment pipelines. With ECR, teams can focus on building and deploying
their
        applications without having to worry about the underlying infrastructure
required to
        host and manage a container registry.

        This scenario walks you through how to perform key operations for this
service.
        Let's get started...

         You have two choices:
            1 - Run the entire program.
            2 - Delete an existing Amazon ECR repository named echo-text (created
from a previous execution of
            this program that did not complete).

        """.trimIndent(),
    )

    while (true) {
        val input = scanner.nextLine()
        if (input.trim { it <= ' ' }.equals("1", ignoreCase = true)) {
            println("Continuing with the program...")
            println("")
            break
        } else if (input.trim { it <= ' ' }.equals("2", ignoreCase = true)) {
            val repoName = "echo-text"
            ecrActions.deleteECRRepository(repoName)
            return
        } else {
            // Handle invalid input.
            println("Invalid input. Please try again.")
        }
    }
```

```
    waitForInputToContinue(scanner)
    println(DASHES)
    println(
        """
        1. Create an ECR repository.

        The first task is to ensure we have a local Docker image named echo-
text.
        If this image exists, then an Amazon ECR repository is created.

        An ECR repository is a private Docker container repository provided
        by Amazon Web Services (AWS). It is a managed service that makes it easy
        to store, manage, and deploy Docker container images.

        """.trimIndent(),
    )

    // Ensure that a local docker image named echo-text exists.
    val doesExist = ecrActions.listLocalImages()
    val repoName: String
    if (!doesExist) {
        println("The local image named echo-text does not exist")
        return
    } else {
        localImageName = "echo-text"
        repoName = "echo-text"
    }

    val repoArn = ecrActions.createECRRepository(repoName).toString()
    println("The ARN of the ECR repository is $repoArn")
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        2. Set an ECR repository policy.

        Setting an ECR repository policy using the `setRepositoryPolicy` function
 is crucial for maintaining
        the security and integrity of your container images. The repository
 policy allows you to
        define specific rules and restrictions for accessing and managing the
 images stored within your ECR
```

```
        repository.

        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    ecrActions.setRepoPolicy(repoName, iamRole)
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        3. Display ECR repository policy.

        Now we will retrieve the ECR policy to ensure it was successfully set.
        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    val policyText = ecrActions.getRepoPolicy(repoName)
    println("Policy Text:")
    println(policyText)
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        4. Retrieve an ECR authorization token.

        You need an authorization token to securely access and interact with the
Amazon ECR registry.
        The `getAuthorizationToken` method of the `EcrAsyncClient` is responsible
for securely accessing
        and interacting with an Amazon ECR repository. This operation is
responsible for obtaining a
        valid authorization token, which is required to authenticate your
requests to the ECR service.

        Without a valid authorization token, you would not be able to perform any
operations on the
        ECR repository, such as pushing, pulling, or managing your Docker images.


        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
```

```
    ecrActions.getAuthToken()
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        5. Get the ECR Repository URI.

        The URI  of an Amazon ECR repository is important. When you want to
deploy a container image to
        a container orchestration platform like Amazon Elastic Kubernetes Service
(EKS)
        or Amazon Elastic Container Service (ECS), you need to specify the full
image URI,
        which includes the ECR repository URI. This allows the container runtime
to pull the
        correct container image from the ECR repository.

        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    val repositoryURI: String? = ecrActions.getRepositoryURI(repoName)
    println("The repository URI is $repositoryURI")
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        6. Set an ECR Lifecycle Policy.

        An ECR Lifecycle Policy is used to manage the lifecycle of Docker images
stored in your ECR repositories.
        These policies allow you to automatically remove old or unused Docker
images from your repositories,
        freeing up storage space and reducing costs.

        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    val pol = ecrActions.setLifeCyclePolicy(repoName)
    println(pol)
    waitForInputToContinue(scanner)

    println(DASHES)
```

```
    println(
        """
        7. Push a docker image to the Amazon ECR Repository.

        The `pushImageCmd()` method pushes a local Docker image to an Amazon ECR
repository.
        It sets up the Docker client by connecting to the local Docker host using
the default port.
        It then retrieves the authorization token for the ECR repository by
making a call to the AWS SDK.

        The method uses the authorization token to create an `AuthConfig` object,
which is used to authenticate
        the Docker client when pushing the image. Finally, the method tags the
Docker image with the specified
        repository name and image tag, and then pushes the image to the ECR
repository using the Docker client.
        If the push operation is successful, the method prints a message
indicating that the image was pushed to ECR.

        """.trimIndent(),
    )

    waitForInputToContinue(scanner)
    ecrActions.pushDockerImage(repoName, localImageName)
    waitForInputToContinue(scanner)

    println(DASHES)
    println("8. Verify if the image is in the ECR Repository.")
    waitForInputToContinue(scanner)
    ecrActions.verifyImage(repoName, localImageName)
    waitForInputToContinue(scanner)

    println(DASHES)
    println("9. As an optional step, you can interact with the image in Amazon
ECR by using the CLI.")
    println("Would you like to view instructions on how to use the CLI to run the
image? (y/n)")
    val ans = scanner.nextLine().trim()
    if (ans.equals("y", true)) {
        val instructions = """
        1. Authenticate with ECR - Before you can pull the image from Amazon ECR,
you need to authenticate with the registry. You can do this using the AWS CLI:
```

```
            aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin $accountId.dkr.ecr.us-east-1.amazonaws.com

        2. Describe the image using this command:

           aws ecr describe-images --repository-name $repoName --image-ids
 imageTag=$localImageName

        3. Run the Docker container and view the output using this command:

           docker run --rm $accountId.dkr.ecr.us-east-1.amazonaws.com/$repoName:
$localImageName
        """
        println(instructions)
    }
    waitForInputToContinue(scanner)

    println(DASHES)
    println("10. Delete the ECR Repository.")
    println(
        """
        If the repository isn't empty, you must either delete the contents of the
 repository
        or use the force option (used in this scenario) to delete the repository
 and have Amazon ECR delete all of its contents
        on your behalf.

        """.trimIndent(),
    )
    println("Would you like to delete the Amazon ECR Repository? (y/n)")
    val delAns = scanner.nextLine().trim { it <= ' ' }
    if (delAns.equals("y", ignoreCase = true)) {
        println("You selected to delete the AWS ECR resources.")
        waitForInputToContinue(scanner)
        ecrActions.deleteECRRepository(repoName)
    }

    println(DASHES)
    println("This concludes the Amazon ECR SDK scenario")
    println(DASHES)
}

private fun waitForInputToContinue(scanner: Scanner) {
    while (true) {
```

```
        println("")
        println("Enter 'c' followed by <ENTER> to continue:")
        val input = scanner.nextLine()
        if (input.trim { it <= ' ' }.equals("c", ignoreCase = true)) {
            println("Continuing with the program...")
            println("")
            break
        } else {
            // Handle invalid input.
            println("Invalid input. Please try again.")
        }
    }
}
```

Amazon ECR SDK メソッドのラッパークラス。

```
import aws.sdk.kotlin.services.ecr.EcrClient
import aws.sdk.kotlin.services.ecr.model.CreateRepositoryRequest
import aws.sdk.kotlin.services.ecr.model.DeleteRepositoryRequest
import aws.sdk.kotlin.services.ecr.model.DescribeImagesRequest
import aws.sdk.kotlin.services.ecr.model.DescribeRepositoriesRequest
import aws.sdk.kotlin.services.ecr.model.EcrException
import aws.sdk.kotlin.services.ecr.model.GetRepositoryPolicyRequest
import aws.sdk.kotlin.services.ecr.model.ImageIdentifier
import aws.sdk.kotlin.services.ecr.model.RepositoryAlreadyExistsException
import aws.sdk.kotlin.services.ecr.model.SetRepositoryPolicyRequest
import aws.sdk.kotlin.services.ecr.model.StartLifecyclePolicyPreviewRequest
import com.github.dockerjava.api.DockerClient
import com.github.dockerjava.api.command.DockerCmdExecFactory
import com.github.dockerjava.api.model.AuthConfig
import com.github.dockerjava.core.DockerClientBuilder
import com.github.dockerjava.netty.NettyDockerCmdExecFactory
import java.io.IOException
import java.util.Base64

class ECRActions {
    private var dockerClient: DockerClient? = null

    private fun getDockerClient(): DockerClient? {
        val osName = System.getProperty("os.name")
        if (osName.startsWith("Windows")) {
            // Make sure Docker Desktop is running.
```

```
            val dockerHost = "tcp://localhost:2375" // Use the Docker Desktop
default port.
            val dockerCmdExecFactory: DockerCmdExecFactory =

NettyDockerCmdExecFactory().withReadTimeout(20000).withConnectTimeout(20000)
            dockerClient =
DockerClientBuilder.getInstance(dockerHost).withDockerCmdExecFactory(dockerCmdExecFactor
        } else {
            dockerClient = DockerClientBuilder.getInstance().build()
        }
        return dockerClient
    }


    /**
     * Sets the lifecycle policy for the specified repository.
     *
     * @param repoName the name of the repository for which to set the lifecycle
policy.
     */
    suspend fun setLifeCyclePolicy(repoName: String): String? {
        val polText =
            """
             {
             "rules": [
                 {
                     "rulePriority": 1,
                     "description": "Expire images older than 14 days",
                     "selection": {
                         "tagStatus": "any",
                         "countType": "sinceImagePushed",
                         "countUnit": "days",
                         "countNumber": 14
                     },
                     "action": {
                         "type": "expire"
                     }
                 }
             ]
             }

            """.trimIndent()
        val lifecyclePolicyPreviewRequest =
            StartLifecyclePolicyPreviewRequest {
```

```
                lifecyclePolicyText = polText
                repositoryName = repoName
            }

        // Execute the request asynchronously.
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val response =
ecrClient.startLifecyclePolicyPreview(lifecyclePolicyPreviewRequest)
            return response.lifecyclePolicyText
        }
    }


    /**
     * Retrieves the repository URI for the specified repository name.
     *
     * @param repoName the name of the repository to retrieve the URI for.
     * @return the repository URI for the specified repository name.
     */
    suspend fun getRepositoryURI(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
cannot be null or empty" }
        val request =
            DescribeRepositoriesRequest {
                repositoryNames = listOf(repoName)
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val describeRepositoriesResponse =
ecrClient.describeRepositories(request)
            if (!describeRepositoriesResponse.repositories?.isEmpty()!!) {
                return
describeRepositoriesResponse?.repositories?.get(0)?.repositoryUri
            } else {
                println("No repositories found for the given name.")
                return ""
            }
        }
    }


    /**
     * Retrieves the authorization token for Amazon Elastic Container Registry
(ECR).
```

```
     *
     */
    suspend fun getAuthToken() {
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            // Retrieve the authorization token for ECR.
            val response = ecrClient.getAuthorizationToken()
            val authorizationData = response.authorizationData?.get(0)
            val token = authorizationData?.authorizationToken
            if (token != null) {
                println("The token was successfully retrieved.")
            }
        }
    }


    /**
     * Gets the repository policy for the specified repository.
     *
     * @param repoName the name of the repository.
     */
    suspend fun getRepoPolicy(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
cannot be null or empty" }

        // Create the request
        val getRepositoryPolicyRequest =
            GetRepositoryPolicyRequest {
                repositoryName = repoName
            }
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val response =
ecrClient.getRepositoryPolicy(getRepositoryPolicyRequest)
            val responseText = response.policyText
            return responseText
        }
    }


    /**
     * Sets the repository policy for the specified ECR repository.
     *
     * @param repoName the name of the ECR repository.
     * @param iamRole the IAM role to be granted access to the repository.
     */
```

```
    suspend fun setRepoPolicy(
        repoName: String?,
        iamRole: String?,
    ) {
        val policyDocumentTemplate =
            """
             {
              "Version":"2012-10-17",
              "Statement" : [ {
                "Sid" : "new statement",
                "Effect" : "Allow",
                "Principal" : {
                  "AWS" : "$iamRole"
                },
                "Action" : "ecr:BatchGetImage"
              } ]
            }

            """.trimIndent()
        val setRepositoryPolicyRequest =
            SetRepositoryPolicyRequest {
                repositoryName = repoName
                policyText = policyDocumentTemplate
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val response =
ecrClient.setRepositoryPolicy(setRepositoryPolicyRequest)
            if (response != null) {
                println("Repository policy set successfully.")
            }
        }
    }


    /**
     * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
     *
     * @param repoName the name of the repository to create.
     * @return the Amazon Resource Name (ARN) of the created repository, or an
empty string if the operation failed.
     * @throws RepositoryAlreadyExistsException if the repository exists.
     * @throws EcrException         if an error occurs while creating the
repository.
```

```
     */
    suspend fun createECRRepository(repoName: String?): String? {
        val request =
            CreateRepositoryRequest {
                repositoryName = repoName
            }

        return try {
            EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
                val response = ecrClient.createRepository(request)
                response.repository?.repositoryArn
            }
        } catch (e: RepositoryAlreadyExistsException) {
            println("Repository already exists: $repoName")
            repoName?.let { getRepoARN(it) }
        } catch (e: EcrException) {
            println("An error occurred: ${e.message}")
            null
        }
    }

    suspend fun getRepoARN(repoName: String): String? {
        // Fetch the existing repository's ARN.
        val describeRequest =
            DescribeRepositoriesRequest {
                repositoryNames = listOf(repoName)
            }
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val describeResponse =
ecrClient.describeRepositories(describeRequest)
            return describeResponse.repositories?.get(0)?.repositoryArn
        }
    }

    fun listLocalImages(): Boolean = try {
        val images = getDockerClient()?.listImagesCmd()?.exec()
        images?.any { image ->
            image.repoTags?.any { tag -> tag.startsWith("echo-text") } ?: false
        } ?: false
    } catch (ex: Exception) {
        println("ERROR: ${ex.message}")
        false
    }
```

```
   /**
    * Pushes a Docker image to an Amazon Elastic Container Registry (ECR)
repository.
    *
    * @param repoName the name of the ECR repository to push the image to.
    * @param imageName the name of the Docker image.
    */
   suspend fun pushDockerImage(
       repoName: String,
       imageName: String,
   ) {
       println("Pushing $imageName to $repoName will take a few seconds")
       val authConfig = getAuthConfig(repoName)

       EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
           val desRequest =
               DescribeRepositoriesRequest {
                   repositoryNames = listOf(repoName)
               }

           val describeRepoResponse = ecrClient.describeRepositories(desRequest)
           val repoData =
               describeRepoResponse.repositories?.firstOrNull
{ it.repositoryName == repoName }
                   ?: throw RuntimeException("Repository not found: $repoName")

           val tagImageCmd = getDockerClient()?.tagImageCmd("$imageName",
"${repoData.repositoryUri}", imageName)
           if (tagImageCmd != null) {
               tagImageCmd.exec()
           }
           val pushImageCmd =
               repoData.repositoryUri?.let {
                   dockerClient?.pushImageCmd(it)
                       // ?.withTag("latest")
                       ?.withAuthConfig(authConfig)
               }

           try {
               if (pushImageCmd != null) {
                   pushImageCmd.start().awaitCompletion()
               }
               println("The $imageName was pushed to Amazon ECR")
```

```
            } catch (e: IOException) {
                throw RuntimeException(e)
            }
        }
    }


    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     */
    suspend fun verifyImage(
        repoName: String?,
        imageTagVal: String?,
    ) {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        require(!(imageTagVal == null || imageTagVal.isEmpty())) { "Image tag
 cannot be null or empty" }

        val imageId =
            ImageIdentifier {
                imageTag = imageTagVal
            }
        val request =
            DescribeImagesRequest {
                repositoryName = repoName
                imageIds = listOf(imageId)
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val describeImagesResponse = ecrClient.describeImages(request)
            if (describeImagesResponse != null && !
describeImagesResponse.imageDetails?.isEmpty()!!) {
                println("Image is present in the repository.")
            } else {
                println("Image is not present in the repository.")
            }
        }
    }
```

```
    /**
     * Deletes an ECR (Elastic Container Registry) repository.
     *
     * @param repoName the name of the repository to delete.
     */
    suspend fun deleteECRRepository(repoName: String) {
        if (repoName.isNullOrEmpty()) {
            throw IllegalArgumentException("Repository name cannot be null or
empty")
        }

        val repositoryRequest =
            DeleteRepositoryRequest {
                force = true
                repositoryName = repoName
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            ecrClient.deleteRepository(repositoryRequest)
            println("You have successfully deleted the $repoName repository")
        }
    }

    // Return an AuthConfig.
    private suspend fun getAuthConfig(repoName: String): AuthConfig {
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            // Retrieve the authorization token for ECR.
            val response = ecrClient.getAuthorizationToken()
            val authorizationData = response.authorizationData?.get(0)
            val token = authorizationData?.authorizationToken
            val decodedToken = String(Base64.getDecoder().decode(token))
            val password = decodedToken.substring(4)

            val request =
                DescribeRepositoriesRequest {
                    repositoryNames = listOf(repoName)
                }

            val descrRepoResponse = ecrClient.describeRepositories(request)
            val repoData = descrRepoResponse.repositories?.firstOrNull
{ it.repositoryName == repoName }
            val registryURL: String =
repoData?.repositoryUri?.split("/")?.get(0) ?: ""
```

```
            return AuthConfig()
                .withUsername("AWS")
                .withPassword(password)
                .withRegistryAddress(registryURL)
        }
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の以下のトピックを参照してください。
  - CreateRepository
  - DeleteRepository
  - DescribeImages
  - DescribeRepositories
  - GetAuthorizationToken
  - GetRepositoryPolicy
  - SetRepositoryPolicy
  - StartLifecyclePolicyPreview

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。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
class ECRGettingStarted:
    """
    A scenario that demonstrates how to use Boto3 to perform basic operations
 using
    Amazon ECR.
```

```
    """

    def __init__(
        self,
        ecr_wrapper: ECRWrapper,
        docker_client: docker.DockerClient,
    ):
        self.ecr_wrapper = ecr_wrapper
        self.docker_client = docker_client
        self.tag = "echo-text"
        self.repository_name = "ecr-basics"
        self.docker_image = None
        self.full_tag_name = None
        self.repository = None

    def run(self, role_arn: str) -> None:
        """
        Runs the scenario.
        """
        print(
            """
The Amazon Elastic Container Registry (ECR) is a fully-managed Docker container
 registry
service provided by AWS. It allows developers and organizations to securely
store, manage, and deploy Docker container images.
ECR provides a simple and scalable way to manage container images throughout
 their lifecycle,
from building and testing to production deployment.

The `ECRWrapper' class is a wrapper for the Boto3 'ecr' client. The 'ecr' client
 provides a set of methods to
programmatically interact with the Amazon ECR service. This allows developers to
automate the storage, retrieval, and management of container images as part of
 their application
deployment pipelines. With ECR, teams can focus on building and deploying their
applications without having to worry about the underlying infrastructure required
 to
host and manage a container registry.

This scenario walks you through how to perform key operations for this service.
Let's get started...
        """
        )
        press_enter_to_continue()
```

```
        print_dashes()
        print(
            f"""
* Create an ECR repository.

An ECR repository is a private Docker container repository provided
by Amazon Web Services (AWS). It is a managed service that makes it easy
to store, manage, and deploy Docker container images.
        """
        )
        print(f"Creating a repository named {self.repository_name}")
        self.repository =
 self.ecr_wrapper.create_repository(self.repository_name)
        print(f"The ARN of the ECR repository is
 {self.repository['repositoryArn']}")
        repository_uri = self.repository["repositoryUri"]
        press_enter_to_continue()
        print_dashes()

        print(
            f"""
* Build a Docker image.

Create a local Docker image if it does not already exist.
A Python Docker client is used to execute Docker commands.
You must have Docker installed and running.
            """
        )
        print(f"Building a docker image from 'docker_files/Dockerfile'")
        self.full_tag_name = f"{repository_uri}:{self.tag}"
        self.docker_image = self.docker_client.images.build(
            path="docker_files", tag=self.full_tag_name
        )[0]
        print(f"Docker image {self.full_tag_name} successfully built.")
        press_enter_to_continue()
        print_dashes()

        if role_arn is None:
            print(
                """
* Because an IAM role ARN was not provided, a role policy will not be set for
 this repository.
            """
            )
```

```
        else:
            print(
                """
* Set an ECR repository policy.

Setting an ECR repository policy using the `setRepositoryPolicy` function is
 crucial for maintaining
the security and integrity of your container images. The repository policy allows
 you to
define specific rules and restrictions for accessing and managing the images
 stored within your ECR
repository.
        """
            )

            self.grant_role_download_access(role_arn)
            print(f"Download access granted to the IAM role ARN {role_arn}")
            press_enter_to_continue()
            print_dashes()

            print(
                """
* Display ECR repository policy.

Now we will retrieve the ECR policy to ensure it was successfully set.
            """
            )

            policy_text =
 self.ecr_wrapper.get_repository_policy(self.repository_name)
            print("Policy Text:")
            print(f"{policy_text}")
            press_enter_to_continue()
            print_dashes()

        print(
            """
* Retrieve an ECR authorization token.

You need an authorization token to securely access and interact with the Amazon
 ECR registry.
The `get_authorization_token` method of the `ecr` client is responsible for
 securely accessing
```

```
and interacting with an Amazon ECR repository. This operation is responsible for
 obtaining a
valid authorization token, which is required to authenticate your requests to the
 ECR service.

Without a valid authorization token, you would not be able to perform any
 operations on the
ECR repository, such as pushing, pulling, or managing your Docker images.
        """
        )

        authorization_token = self.ecr_wrapper.get_authorization_token()
        print("Authorization token retrieved.")
        press_enter_to_continue()
        print_dashes()
        print(
            """
* Get the ECR Repository URI.

The URI  of an Amazon ECR repository is important. When you want to deploy a
 container image to
a container orchestration platform like Amazon Elastic Kubernetes Service (EKS)
or Amazon Elastic Container Service (ECS), you need to specify the full image
 URI,
which includes the ECR repository URI. This allows the container runtime to pull
 the
correct container image from the ECR repository.
        """
        )
        repository_descriptions = self.ecr_wrapper.describe_repositories(
            [self.repository_name]
        )
        repository_uri = repository_descriptions[0]["repositoryUri"]
        print(f"Repository URI found: {repository_uri}")
        press_enter_to_continue()
        print_dashes()

        print(
            """
* Set an ECR Lifecycle Policy.

An ECR Lifecycle Policy is used to manage the lifecycle of Docker images stored
 in your ECR repositories.
```

```
These policies allow you to automatically remove old or unused Docker images from
 your repositories,
freeing up storage space and reducing costs.

This example policy helps to maintain the size and efficiency of the container
 registry
by automatically removing older and potentially unused images, ensuring that the
storage is optimized and the registry remains up-to-date.
            """
        )
        press_enter_to_continue()
        self.put_expiration_policy()
        print(f"An expiration policy was added to the repository.")
        print_dashes()

        print(
            """
* Push a docker image to the Amazon ECR Repository.

The Docker client uses the authorization token is used to authenticate the when
 pushing the image to the
ECR repository.
        """
        )
        decoded_authorization =
 base64.b64decode(authorization_token).decode("utf-8")
        username, password = decoded_authorization.split(":")

        resp = self.docker_client.api.push(
            repository=repository_uri,
            auth_config={"username": username, "password": password},
            tag=self.tag,
            stream=True,
            decode=True,
        )
        for line in resp:
            print(line)

        print_dashes()

        print("* Verify if the image is in the ECR Repository.")
        image_descriptions = self.ecr_wrapper.describe_images(
            self.repository_name, [self.tag]
        )
```

```
        if len(image_descriptions) > 0:
            print("Image found in ECR Repository.")
        else:
            print("Image not found in ECR Repository.")
        press_enter_to_continue()
        print_dashes()

        print(
            "* As an optional step, you can interact with the image in Amazon ECR
 by using the CLI."
        )
        if q.ask(
            "Would you like to view instructions on how to use the CLI to run the
 image? (y/n)",
            q.is_yesno,
        ):
            print(
                f"""
1. Authenticate with ECR - Before you can pull the image from Amazon ECR, you
 need to authenticate with the registry. You can do this using the AWS CLI:

    aws ecr get-login-password --region us-east-1 | docker login --username AWS
 --password-stdin {repository_uri.split("/")[0]}

2. Describe the image using this command:

   aws ecr describe-images --repository-name {self.repository_name} --image-ids
 imageTag={self.tag}

3. Run the Docker container and view the output using this command:

   docker run --rm {self.full_tag_name}
"""
            )

        self.cleanup(True)

    def cleanup(self, ask: bool):
        """
        Deletes the resources created in this scenario.
        :param ask: If True, prompts the user to confirm before deleting the
 resources.
        """
        if self.repository is not None and (
```

```
            not ask
            or q.ask(
                f"Would you like to delete the ECR repository
'{self.repository_name}? (y/n) "
            )
        ):
            print(f"Deleting the ECR repository '{self.repository_name}'.")
            self.ecr_wrapper.delete_repository(self.repository_name)

        if self.full_tag_name is not None and (
            not ask
            or q.ask(
                f"Would you like to delete the local Docker image
'{self.full_tag_name}? (y/n) "
            )
        ):
            print(f"Deleting the docker image '{self.full_tag_name}'.")
            self.docker_client.images.remove(self.full_tag_name)

    def grant_role_download_access(self, role_arn: str):
        """
        Grants the specified role access to download images from the ECR
repository.

        :param role_arn: The ARN of the role to grant access to.
        """
        policy_json = {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Sid": "AllowDownload",
                    "Effect": "Allow",
                    "Principal": {"AWS": role_arn},
                    "Action": ["ecr:BatchGetImage"],
                }
            ],
        }

        self.ecr_wrapper.set_repository_policy(
            self.repository_name, json.dumps(policy_json)
        )


    def put_expiration_policy(self):
```

```
        """
        Puts an expiration policy on the ECR repository.
        """
        policy_json = {
            "rules": [
                {
                    "rulePriority": 1,
                    "description": "Expire images older than 14 days",
                    "selection": {
                        "tagStatus": "any",
                        "countType": "sinceImagePushed",
                        "countUnit": "days",
                        "countNumber": 14,
                    },
                    "action": {"type": "expire"},
                }
            ]
        }

        self.ecr_wrapper.put_lifecycle_policy(
            self.repository_name, json.dumps(policy_json)
        )



if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Run Amazon ECR getting started scenario."
    )
    parser.add_argument(
        "--iam-role-arn",
        type=str,
        default=None,
        help="an optional IAM role ARN that will be granted access to download
 images from a repository.",
        required=False,
    )
    parser.add_argument(
        "--no-art",
        action="store_true",
        help="accessibility setting that suppresses art in the console output.",
    )
    args = parser.parse_args()
    no_art = args.no_art
```

```
    iam_role_arn = args.iam_role_arn
    demo = None
    a_docker_client = None
    try:
        a_docker_client = docker.from_env()
        if not a_docker_client.ping():
            raise docker.errors.DockerException("Docker is not running.")
    except docker.errors.DockerException as err:
        logging.error(
            """
        The Python Docker client could not be created.
        Do you have Docker installed and running?
        Here is the error message:
        %s
        """,
            err,
        )
        sys.exit("Error with Docker.")
    try:
        an_ecr_wrapper = ECRWrapper.from_client()
        demo = ECRGettingStarted(an_ecr_wrapper, a_docker_client)
        demo.run(iam_role_arn)

    except Exception as exception:
        logging.exception("Something went wrong with the demo!")
        if demo is not None:
            demo.cleanup(False)
```

Amazon ECR アクションをラップする ECRWrapper クラス。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
```

```
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def create_repository(self, repository_name: str) -> dict[str, any]:
        """
        Creates an ECR repository.

        :param repository_name: The name of the repository to create.
        :return: A dictionary of the created repository.
        """
        try:
            response =
self.ecr_client.create_repository(repositoryName=repository_name)
            return response["repository"]
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"RepositoryAlreadyExistsException":
                print(f"Repository {repository_name} already exists.")
                response = self.ecr_client.describe_repositories(
                    repositoryNames=[repository_name]
                )
                return self.describe_repositories([repository_name])[0]
            else:
                logger.error(
                    "Error creating repository %s. Here's why %s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise


    def delete_repository(self, repository_name: str):
        """
        Deletes an ECR repository.

        :param repository_name: The name of the repository to delete.
        """
        try:
            self.ecr_client.delete_repository(
                repositoryName=repository_name, force=True
            )
            print(f"Deleted repository {repository_name}.")
```

```
        except ClientError as err:
            logger.error(
                "Couldn't delete repository %s.. Here's why %s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise


    def set_repository_policy(self, repository_name: str, policy_text: str):
        """
        Sets the policy for an ECR repository.

        :param repository_name: The name of the repository to set the policy for.
        :param policy_text: The policy text to set.
        """
        try:
            self.ecr_client.set_repository_policy(
                repositoryName=repository_name, policyText=policy_text
            )
            print(f"Set repository policy for repository {repository_name}.")
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
                    "Couldn't set repository policy for repository %s. Here's why
%s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise


    def get_repository_policy(self, repository_name: str) -> str:
        """
        Gets the policy for an ECR repository.

        :param repository_name: The name of the repository to get the policy for.
        :return: The policy text.
        """
        try:
```

```
            response = self.ecr_client.get_repository_policy(
                repositoryName=repository_name
            )
            return response["policyText"]
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
                    "Couldn't get repository policy for repository %s. Here's why
%s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise


    def get_authorization_token(self) -> str:
        """
        Gets an authorization token for an ECR repository.

        :return: The authorization token.
        """
        try:
            response = self.ecr_client.get_authorization_token()
            return response["authorizationData"][0]["authorizationToken"]
        except ClientError as err:
            logger.error(
                "Couldn't get authorization token. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise


    def describe_repositories(self, repository_names: list[str]) -> list[dict]:
        """
        Describes ECR repositories.

        :param repository_names: The names of the repositories to describe.
        :return: The list of repository descriptions.
        """
        try:
```

```
            response = self.ecr_client.describe_repositories(
                repositoryNames=repository_names
            )
            return response["repositories"]
        except ClientError as err:
            logger.error(
                "Couldn't describe repositories. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise


    def put_lifecycle_policy(self, repository_name: str, lifecycle_policy_text:
str):
        """
        Puts a lifecycle policy for an ECR repository.

        :param repository_name: The name of the repository to put the lifecycle
policy for.
        :param lifecycle_policy_text: The lifecycle policy text to put.
        """
        try:
            self.ecr_client.put_lifecycle_policy(
                repositoryName=repository_name,
                lifecyclePolicyText=lifecycle_policy_text,
            )
            print(f"Put lifecycle policy for repository {repository_name}.")
        except ClientError as err:
            logger.error(
                "Couldn't put lifecycle policy for repository %s. Here's why %s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise


    def describe_images(
        self, repository_name: str, image_ids: list[str] = None
    ) -> list[dict]:
        """
        Describes ECR images.

        :param repository_name: The name of the repository to describe images
for.
```

```
        :param image_ids: The optional IDs of images to describe.
        :return: The list of image descriptions.
        """
        try:
            params = {
                "repositoryName": repository_name,
            }
            if image_ids is not None:
                params["imageIds"] = [{"imageTag": tag} for tag in image_ids]

            paginator = self.ecr_client.get_paginator("describe_images")
            image_descriptions = []
            for page in paginator.paginate(**params):
                image_descriptions.extend(page["imageDetails"])
            return image_descriptions
        except ClientError as err:
            logger.error(
                "Couldn't describe images. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- 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。
  - CreateRepository
  - DeleteRepository
  - DescribeImages
  - DescribeRepositories
  - GetAuthorizationToken
  - GetRepositoryPolicy
  - SetRepositoryPolicy
  - StartLifecyclePolicyPreview

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 Amazon ECR 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 SDKsアクション

次のコード例は、 AWS SDKs を使用して個々の Amazon ECR 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。

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。完全なリストについては、「Amazon Elastic Container Registry API Reference」を参照してください。

例

- AWS SDK または CLI CreateRepositoryで を使用する
- AWS SDK または CLI DeleteRepositoryで を使用する
- AWS SDK または CLI DescribeImagesで を使用する
- AWS SDK または CLI DescribeRepositoriesで を使用する
- AWS SDK または CLI GetAuthorizationTokenで を使用する
- AWS SDK または CLI GetRepositoryPolicyで を使用する
- AWS SDK または CLI ListImagesで を使用する
- AWS SDK PushImageCmdで を使用する
- AWS SDK または CLI PutLifeCyclePolicyで を使用する
- AWS SDK または CLI SetRepositoryPolicyで を使用する
- AWS SDK または CLI StartLifecyclePolicyPreviewで を使用する

AWS SDK または CLI **CreateRepository**で を使用する

次のサンプルコードは、`CreateRepository` 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。

- 基本を学ぶ

CLI

AWS CLI

例 1: リポジトリを作成する場合

次の create-repository の例では、アカウントのデフォルトレジストリ内の指定された名前空間にリポジトリを作成します。

```
aws ecr create-repository \
    --repository-name project-a/sample-repo
```

出力:

```
{
    "repository": {
        "registryId": "123456789012",
        "repositoryName": "project-a/sample-repo",
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/project-
a/sample-repo"
    }
}
```

詳細については、「Amazon ECR ユーザーガイド」の「リポジトリの作成」を参照してください。

例 2: イメージタグのイミュータビリティが設定されたリポジトリを作成する場合

次の create-repository の例では、アカウントのデフォルトレジストリ内にタグのイミュータビリティが設定されたリポジトリを作成します。

```
aws ecr create-repository \
    --repository-name project-a/sample-repo \
    --image-tag-mutability IMMUTABLE
```

出力:

```
{
    "repository": {
        "registryId": "123456789012",
        "repositoryName": "project-a/sample-repo",
```

```
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/project-
a/sample-repo",
        "imageTagMutability": "IMMUTABLE"
    }
}
```

詳細については、「Amazon ECR ユーザーガイド」の「イメージタグの変更可能性」を参照してください。

例 3: スキャン設定が設定されたリポジトリを作成する場合

次の `create-repository` の例では、アカウントのデフォルトレジストリ内でイメージプッシュに対して脆弱性スキャンを実行するように設定されたリポジトリを作成します。

```
aws ecr create-repository \
    --repository-name project-a/sample-repo \
    --image-scanning-configuration scanOnPush=true
```

出力:

```
{
    "repository": {
        "registryId": "123456789012",
        "repositoryName": "project-a/sample-repo",
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/project-
a/sample-repo",
        "imageScanningConfiguration": {
            "scanOnPush": true
        }
    }
}
```

詳細については、「Amazon ECR ユーザーガイド」の「イメージスキャン」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「CreateRepository」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。

```
   /**
    * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
    *
    * @param repoName the name of the repository to create.
    * @return the Amazon Resource Name (ARN) of the created repository, or an
empty string if the operation failed.
    * @throws IllegalArgumentException     If repository name is invalid.
    * @throws RuntimeException             if an error occurs while creating the
repository.
    */
   public String createECRRepository(String repoName) {
       if (repoName == null || repoName.isEmpty()) {
           throw new IllegalArgumentException("Repository name cannot be null or
empty");
       }

       CreateRepositoryRequest request = CreateRepositoryRequest.builder()
           .repositoryName(repoName)
           .build();

       CompletableFuture<CreateRepositoryResponse> response =
getAsyncClient().createRepository(request);
       try {
           CreateRepositoryResponse result = response.join();
           if (result != null) {
               System.out.println("The " + repoName + " repository was created
successfully.");
               return result.repository().repositoryArn();
           } else {
               throw new RuntimeException("Unexpected response type");
           }
       } catch (CompletionException e) {
```

```
            Throwable cause = e.getCause();
            if (cause instanceof EcrException ex) {
                if
("RepositoryAlreadyExistsException".equals(ex.awsErrorDetails().errorCode())) {
                    System.out.println("The Amazon ECR repository already exists,
moving on...");
                    DescribeRepositoriesRequest describeRequest =
DescribeRepositoriesRequest.builder()
                        .repositoryNames(repoName)
                        .build();
                    DescribeRepositoriesResponse describeResponse =
getAsyncClient().describeRepositories(describeRequest).join();
                    return
describeResponse.repositories().get(0).repositoryArn();
                } else {
                    throw new RuntimeException(ex);
                }
            } else {
                throw new RuntimeException(e);
            }
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CreateRepository」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。

```
/**
 * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
 *
```

```
    * @param repoName the name of the repository to create.
    * @return the Amazon Resource Name (ARN) of the created repository, or an
empty string if the operation failed.
    * @throws RepositoryAlreadyExistsException if the repository exists.
    * @throws EcrException         if an error occurs while creating the
repository.
    */
   suspend fun createECRRepository(repoName: String?): String? {
       val request =
           CreateRepositoryRequest {
               repositoryName = repoName
           }

       return try {
           EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
               val response = ecrClient.createRepository(request)
               response.repository?.repositoryArn
           }
       } catch (e: RepositoryAlreadyExistsException) {
           println("Repository already exists: $repoName")
           repoName?.let { getRepoARN(it) }
       } catch (e: EcrException) {
           println("An error occurred: ${e.message}")
           null
       }
   }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「CreateRepository」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def create_repository(self, repository_name: str) -> dict[str, any]:
        """
        Creates an ECR repository.

        :param repository_name: The name of the repository to create.
        :return: A dictionary of the created repository.
        """
        try:
            response =
 self.ecr_client.create_repository(repositoryName=repository_name)
            return response["repository"]
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "RepositoryAlreadyExistsException":
                print(f"Repository {repository_name} already exists.")
                response = self.ecr_client.describe_repositories(
                    repositoryNames=[repository_name]
                )
                return self.describe_repositories([repository_name])[0]
            else:
                logger.error(
                    "Error creating repository %s. Here's why %s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「CreateRepository」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_repository_name = 'my-repository'
        oo_result = lo_ecr->createrepository(
          iv_repositoryname = iv_repository_name ).
        DATA(lv_repository_uri) = oo_result->get_repository( )-
>get_repositoryuri( ).
        MESSAGE |Repository created with URI: { lv_repository_uri }| TYPE 'I'.
      CATCH /aws1/cx_ecrrepositoryalrexex.
        " If repository already exists, retrieve it
        DATA lt_repo_names TYPE /aws1/
cl_ecrrepositorynamels00=>tt_repositorynamelist.
        APPEND NEW /aws1/cl_ecrrepositorynamels00( iv_value =
 iv_repository_name ) TO lt_repo_names.
        DATA(lo_describe_result) = lo_ecr-
>describerepositories( it_repositorynames = lt_repo_names ).
        DATA(lt_repos) = lo_describe_result->get_repositories( ).
        IF lines( lt_repos ) > 0.
          READ TABLE lt_repos INDEX 1 INTO DATA(lo_repo).
          oo_result = NEW /aws1/cl_ecrcrerepositoryrsp( io_repository =
 lo_repo ).
          MESSAGE |Repository { iv_repository_name } already exists.| TYPE 'I'.
        ENDIF.
    ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスの「CreateRepository」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 Amazon ECR 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 **DeleteRepository**で を使用する

次のサンプルコードは、DeleteRepository 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。

- 基本を学ぶ

CLI

AWS CLI

リポジトリを削除するには

次のコマンドフォース delete-repository の例では、アカウントのデフォルトレジストリ内の指定されたリポジトリを削除します。リポジトリにイメージが含まれている場合は、--force フラグが必要です。

```
aws ecr delete-repository \
    --repository-name ubuntu \
    --force
```

出力:

```
{
    "repository": {
        "registryId": "123456789012",
        "repositoryName": "ubuntu",
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/ubuntu"
    }
}
```

詳細については、「Amazon ECR ユーザーガイド」の「リポジトリの削除」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteRepository」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。

```
   /**
    * Deletes an ECR (Elastic Container Registry) repository.
    *
    * @param repoName the name of the repository to delete.
    * @throws IllegalArgumentException if the repository name is null or empty.
    * @throws EcrException if there is an error deleting the repository.
    * @throws RuntimeException if an unexpected error occurs during the deletion
process.
    */
   public void deleteECRRepository(String repoName) {
       if (repoName == null || repoName.isEmpty()) {
           throw new IllegalArgumentException("Repository name cannot be null or
empty");
       }

       DeleteRepositoryRequest repositoryRequest =
DeleteRepositoryRequest.builder()
           .force(true)
           .repositoryName(repoName)
           .build();

       CompletableFuture<DeleteRepositoryResponse> response =
getAsyncClient().deleteRepository(repositoryRequest);
       response.whenComplete((deleteRepositoryResponse, ex) -> {
           if (deleteRepositoryResponse != null) {
```

```
                System.out.println("You have successfully deleted the " +
repoName + " repository");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
                    throw new RuntimeException("Unexpected error: " +
cause.getMessage(), cause);
                }
            }
        });

        // Wait for the CompletableFuture to complete
        response.join();
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteRepository」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。

```
    /**
     * Deletes an ECR (Elastic Container Registry) repository.
     *
     * @param repoName the name of the repository to delete.
     */
    suspend fun deleteECRRepository(repoName: String) {
        if (repoName.isNullOrEmpty()) {
            throw IllegalArgumentException("Repository name cannot be null or
empty")
```

```
        }

        val repositoryRequest =
            DeleteRepositoryRequest {
                force = true
                repositoryName = repoName
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            ecrClient.deleteRepository(repositoryRequest)
            println("You have successfully deleted the $repoName repository")
        }
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DeleteRepository」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
```

```
        return cls(ecr_client)


    def delete_repository(self, repository_name: str):
        """
        Deletes an ECR repository.

        :param repository_name: The name of the repository to delete.
        """
        try:
            self.ecr_client.delete_repository(
                repositoryName=repository_name, force=True
            )
            print(f"Deleted repository {repository_name}.")
        except ClientError as err:
            logger.error(
                "Couldn't delete repository %s.. Here's why %s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DeleteRepository」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_repository_name = 'my-repository'
        lo_ecr->deleterepository(
          iv_repositoryname = iv_repository_name
```

```
          iv_force = abap_true ).
        MESSAGE |Repository { iv_repository_name } deleted.| TYPE 'I'.
      CATCH /aws1/cx_ecrrepositorynotfndex.
        MESSAGE 'Repository not found.' TYPE 'I'.
    ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスのDeleteRepository」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 Amazon ECR 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 **DescribeImages**で を使用する

次のサンプルコードは、DescribeImages 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。

- 基本を学ぶ

CLI

AWS CLI

リポジトリ内のイメージを記述する場合

次の のdescribe-images例では、cluster-autoscalerリポジトリ内のイメージの詳細をタグ とともに表示しますv1.13.6。

```
aws ecr describe-images \
    --repository-name cluster-autoscaler \
    --image-ids imageTag=v1.13.6
```

出力:

```
{
    "imageDetails": [
        {
```

```
            "registryId": "012345678910",
            "repositoryName": "cluster-autoscaler",
            "imageDigest":
 "sha256:4a1c6567c38904384ebc64e35b7eeddd8451110c299e3368d2210066487d97e5",
            "imageTags": [
                "v1.13.6"
            ],
            "imageSizeInBytes": 48318255,
            "imagePushedAt": 1565128275.0
        }
    ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「DescribeImages」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。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
(Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     * @throws EcrException             if there is an error retrieving the image
information from Amazon ECR.
     * @throws CompletionException      if the asynchronous operation completes
exceptionally.
     */
    public void verifyImage(String repositoryName, String imageTag) {
        DescribeImagesRequest request = DescribeImagesRequest.builder()
            .repositoryName(repositoryName)
            .imageIds(ImageIdentifier.builder().imageTag(imageTag).build())
```

```
            .build();

        CompletableFuture<DescribeImagesResponse> response =
 getAsyncClient().describeImages(request);
        response.whenComplete((describeImagesResponse, ex) -> {
            if (ex != null) {
                if (ex instanceof CompletionException) {
                    Throwable cause = ex.getCause();
                    if (cause instanceof EcrException) {
                        throw (EcrException) cause;
                    } else {
                        throw new RuntimeException("Unexpected error: " +
 cause.getMessage(), cause);
                    }
                } else {
                    throw new RuntimeException("Unexpected error: " +
 ex.getCause());
                }
            } else if (describeImagesResponse != null &&
describeImagesResponse.imageDetails().isEmpty()) {
                System.out.println("Image is present in the repository.");
            } else {
                System.out.println("Image is not present in the repository.");
            }
        });

        // Wait for the CompletableFuture to complete.
        response.join();
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DescribeImages」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。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     */
    suspend fun verifyImage(
        repoName: String?,
        imageTagVal: String?,
    ) {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        require(!(imageTagVal == null || imageTagVal.isEmpty())) { "Image tag
 cannot be null or empty" }

        val imageId =
            ImageIdentifier {
                imageTag = imageTagVal
            }
        val request =
            DescribeImagesRequest {
                repositoryName = repoName
                imageIds = listOf(imageId)
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val describeImagesResponse = ecrClient.describeImages(request)
            if (describeImagesResponse != null && !
describeImagesResponse.imageDetails?.isEmpty()!!) {
                println("Image is present in the repository.")
```

```
            } else {
                println("Image is not present in the repository.")
            }
        }
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DescribeImages」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def describe_images(
        self, repository_name: str, image_ids: list[str] = None
    ) -> list[dict]:
        """
        Describes ECR images.
```

```
        :param repository_name: The name of the repository to describe images
for.
        :param image_ids: The optional IDs of images to describe.
        :return: The list of image descriptions.
        """
        try:
            params = {
                "repositoryName": repository_name,
            }
            if image_ids is not None:
                params["imageIds"] = [{"imageTag": tag} for tag in image_ids]

            paginator = self.ecr_client.get_paginator("describe_images")
            image_descriptions = []
            for page in paginator.paginate(**params):
                image_descriptions.extend(page["imageDetails"])
            return image_descriptions
        except ClientError as err:
            logger.error(
                "Couldn't describe images. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DescribeImages」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_repository_name = 'my-repository'
        " it_image_ids = VALUE #( ( NEW /aws1/cl_ecrimageidentifier( iv_imagetag
= 'latest' ) ) )
        IF it_image_ids IS NOT INITIAL.
          oo_result = lo_ecr->describeimages(
            iv_repositoryname = iv_repository_name
            it_imageids = it_image_ids ).
        ELSE.
          oo_result = lo_ecr->describeimages(
            iv_repositoryname = iv_repository_name ).
        ENDIF.
        DATA(lt_image_details) = oo_result->get_imagedetails( ).
        MESSAGE |Found { lines( lt_image_details ) } images in repository.| TYPE
'I'.
      CATCH /aws1/cx_ecrrepositorynotfndex.
        MESSAGE 'Repository not found.' TYPE 'I'.
      CATCH /aws1/cx_ecrimagenotfoundex.
        MESSAGE 'Image not found.' TYPE 'I'.
      CATCH /aws1/cx_ecrinvalidparameterex.
        MESSAGE 'Invalid parameter provided.' TYPE 'I'.
    ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスの「DescribeImages」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 Amazon ECR 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 **DescribeRepositories**で を使用する

次のサンプルコードは、DescribeRepositories 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。

- 基本を学ぶ

CLI

AWS CLI

レジストリ内のリポジトリを記述するには

この例は、アカウントのデフォルトレジストリ内のリポジトリを記述します。

コマンド:

```
aws ecr describe-repositories
```

出力:

```
{
    "repositories": [
        {
            "registryId": "012345678910",
            "repositoryName": "ubuntu",
            "repositoryArn": "arn:aws:ecr:us-west-2:012345678910:repository/
ubuntu"
        },
        {
            "registryId": "012345678910",
            "repositoryName": "test",
            "repositoryArn": "arn:aws:ecr:us-west-2:012345678910:repository/test"
        }
    ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「DescribeRepositories」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。

```
   /**
    * Retrieves the repository URI for the specified repository name.
    *
    * @param repoName the name of the repository to retrieve the URI for.
    * @return the repository URI for the specified repository name.
    * @throws EcrException        if there is an error retrieving the repository
information.
    * @throws CompletionException if the asynchronous operation completes
exceptionally.
    */
   public void getRepositoryURI(String repoName) {
       DescribeRepositoriesRequest request =
DescribeRepositoriesRequest.builder()
           .repositoryNames(repoName)
           .build();

       CompletableFuture<DescribeRepositoriesResponse> response =
getAsyncClient().describeRepositories(request);
       response.whenComplete((describeRepositoriesResponse, ex) -> {
           if (ex != null) {
               Throwable cause = ex.getCause();
               if (cause instanceof InterruptedException) {
                   Thread.currentThread().interrupt();
                   String errorMessage = "Thread interrupted while waiting for
asynchronous operation: " + cause.getMessage();
                   throw new RuntimeException(errorMessage, cause);
               } else if (cause instanceof EcrException) {
                   throw (EcrException) cause;
               } else {
                   String errorMessage = "Unexpected error: " +
cause.getMessage();
                   throw new RuntimeException(errorMessage, cause);
               }
           } else {
               if (describeRepositoriesResponse != null) {
                   if (!describeRepositoriesResponse.repositories().isEmpty()) {
                       String repositoryUri =
describeRepositoriesResponse.repositories().get(0).repositoryUri();
                       System.out.println("Repository URI found: " +
repositoryUri);
                   } else {
                       System.out.println("No repositories found for the given
name.");
```

```
                    }
                } else {
                    System.err.println("No response received from
describeRepositories.");
                }
            }
        });
        response.join();
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeRepositories」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。

```
    /**
     * Retrieves the repository URI for the specified repository name.
     *
     * @param repoName the name of the repository to retrieve the URI for.
     * @return the repository URI for the specified repository name.
     */
    suspend fun getRepositoryURI(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
cannot be null or empty" }
        val request =
            DescribeRepositoriesRequest {
                repositoryNames = listOf(repoName)
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
```

```
            val describeRepositoriesResponse =
ecrClient.describeRepositories(request)
            if (!describeRepositoriesResponse.repositories?.isEmpty()!!) {
                return
describeRepositoriesResponse?.repositories?.get(0)?.repositoryUri
            } else {
                println("No repositories found for the given name.")
                return ""
            }
        }
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DescribeRepositories」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)
```

```
    def describe_repositories(self, repository_names: list[str]) -> list[dict]:
        """
        Describes ECR repositories.

        :param repository_names: The names of the repositories to describe.
        :return: The list of repository descriptions.
        """
        try:
            response = self.ecr_client.describe_repositories(
                repositoryNames=repository_names
            )
            return response["repositories"]
        except ClientError as err:
            logger.error(
                "Couldn't describe repositories. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DescribeRepositories」を参照してください。

Rust

SDK for Rust

> **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。

```
async fn show_repos(client: &aws_sdk_ecr::Client) -> Result<(),
 aws_sdk_ecr::Error> {
    let rsp = client.describe_repositories().send().await?;

    let repos = rsp.repositories();
```

```
    println!("Found {} repositories:", repos.len());

    for repo in repos {
        println!("  ARN:  {}", repo.repository_arn().unwrap());
        println!("  Name: {}", repo.repository_name().unwrap());
    }

    Ok(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「DescribeRepositories」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.
        " it_repository_names = VALUE #( ( NEW /aws1/
cl_ecrrepositorynamels00( iv_value = 'my-repository' ) ) )
        oo_result = lo_ecr->describerepositories(
          it_repositorynames = it_repository_names ).
        DATA(lt_repositories) = oo_result->get_repositories( ).
        MESSAGE |Found { lines( lt_repositories ) } repositories.| TYPE 'I'.
      CATCH /aws1/cx_ecrrepositorynotfndex.
        MESSAGE 'Repository not found.' TYPE 'I'.
    ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスの「DescribeRepositories」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 Amazon ECR 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 **GetAuthorizationToken**で を使用する

次のサンプルコードは、GetAuthorizationToken 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。

- 基本を学ぶ

CLI

AWS CLI

デフォルトレジストリ用の認可トークンを取得する場合

次のコマンド get-authorization-token の例では、デフォルトレジストリ用の認可トークンを取得します。

```
aws ecr get-authorization-token
```

出力:

```
{
    "authorizationData": [
        {
            "authorizationToken": "QVdTOkN...",
            "expiresAt": 1448875853.241,
            "proxyEndpoint": "https://123456789012.dkr.ecr.us-
west-2.amazonaws.com"
        }
    ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetAuthorizationToken」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。

```
   /**
    * Retrieves the authorization token for Amazon Elastic Container Registry
(ECR).
    * This method makes an asynchronous call to the ECR client to retrieve the
authorization token.
    * If the operation is successful, the method prints the token to the
console.
    * If an exception occurs, the method handles the exception and prints the
error message.
    *
    * @throws EcrException     if there is an error retrieving the authorization
token from ECR.
    * @throws RuntimeException if there is an unexpected error during the
operation.
    */
   public void getAuthToken() {
       CompletableFuture<GetAuthorizationTokenResponse> response =
getAsyncClient().getAuthorizationToken();
       response.whenComplete((authorizationTokenResponse, ex) -> {
           if (authorizationTokenResponse != null) {
               AuthorizationData authorizationData =
authorizationTokenResponse.authorizationData().get(0);
               String token = authorizationData.authorizationToken();
               if (!token.isEmpty()) {
                   System.out.println("The token was successfully retrieved.");
               }
           } else {
               if (ex.getCause() instanceof EcrException) {
                   throw (EcrException) ex.getCause();
               } else {
                   String errorMessage = "Unexpected error occurred: " +
ex.getMessage();
```

```
                    throw new RuntimeException(errorMessage, ex); // Rethrow the
exception
                }
            }
        });
        response.join();
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetAuthorizationToken」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。

```
    /**
     * Retrieves the authorization token for Amazon Elastic Container Registry
(ECR).
     *
     */
    suspend fun getAuthToken() {
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            // Retrieve the authorization token for ECR.
            val response = ecrClient.getAuthorizationToken()
            val authorizationData = response.authorizationData?.get(0)
            val token = authorizationData?.authorizationToken
            if (token != null) {
                println("The token was successfully retrieved.")
            }
        }
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetAuthorizationToken」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def get_authorization_token(self) -> str:
        """
        Gets an authorization token for an ECR repository.

        :return: The authorization token.
        """
        try:
            response = self.ecr_client.get_authorization_token()
            return response["authorizationData"][0]["authorizationToken"]
        except ClientError as err:
            logger.error(
                "Couldn't get authorization token. Here's why %s",
```

```
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「GetAuthorizationToken」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_ecr->getauthorizationtoken( ).
    DATA(lt_auth_data) = oo_result->get_authorizationdata( ).
    IF lines( lt_auth_data ) > 0.
      READ TABLE lt_auth_data INDEX 1 INTO DATA(lo_auth_data).
      DATA(lv_token) = lo_auth_data->get_authorizationtoken( ).
      MESSAGE 'Authorization token retrieved.' TYPE 'I'.
    ENDIF.
  CATCH /aws1/cx_ecrserverexception.
    MESSAGE 'Server exception occurred.' TYPE 'I'.
ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスのGetAuthorizationToken」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 Amazon ECR 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 **GetRepositoryPolicy**で を使用する

次のサンプルコードは、GetRepositoryPolicy 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。

- 基本を学ぶ

CLI

AWS CLI

リポジトリのリポジトリポリシーを取得する場合

次の get-repository-policy の例では、cluster-autoscaler リポジトリのリポジトリポリシーの詳細を表示します。

```
aws ecr get-repository-policy \
    --repository-name cluster-autoscaler
```

出力:

```
{
    "registryId": "012345678910",
    "repositoryName": "cluster-autoscaler",
    "policyText": "{\n  \"Version\" : \"2008-10-17\",\n  \"Statement\" :
 [ {\n    \"Sid\" : \"allow public pull\",\n    \"Effect\" : \"Allow\",\n
 \"Principal\" : \"*\",\n    \"Action\" : [ \"ecr:BatchCheckLayerAvailability\",
 \"ecr:BatchGetImage\", \"ecr:GetDownloadUrlForLayer\" ]\n  } ]\n}"
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetRepositoryPolicy」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。

```
    /**
     * Gets the repository policy for the specified repository.
     *
     * @param repoName the name of the repository.
     * @throws EcrException if an AWS error occurs while getting the repository
policy.
     */
    public String getRepoPolicy(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
empty");
        }

        GetRepositoryPolicyRequest getRepositoryPolicyRequest =
GetRepositoryPolicyRequest.builder()
            .repositoryName(repoName)
            .build();

        CompletableFuture<GetRepositoryPolicyResponse> response =
getAsyncClient().getRepositoryPolicy(getRepositoryPolicyRequest);
        response.whenComplete((resp, ex) -> {
            if (resp != null) {
                System.out.println("Repository policy retrieved successfully.");
            } else {
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
ex.getMessage();
                    throw new RuntimeException(errorMessage, ex);
                }
            }
```

```
        });

        GetRepositoryPolicyResponse result = response.join();
        return result != null ? result.policyText() : null;
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetRepositoryPolicy」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。

```
    /**
     * Gets the repository policy for the specified repository.
     *
     * @param repoName the name of the repository.
     */
    suspend fun getRepoPolicy(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
cannot be null or empty" }

        // Create the request
        val getRepositoryPolicyRequest =
            GetRepositoryPolicyRequest {
                repositoryName = repoName
            }
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val response =
ecrClient.getRepositoryPolicy(getRepositoryPolicyRequest)
            val responseText = response.policyText
            return responseText
        }
```

```
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetRepositoryPolicy」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def get_repository_policy(self, repository_name: str) -> str:
        """
        Gets the policy for an ECR repository.

        :param repository_name: The name of the repository to get the policy for.
        :return: The policy text.
        """
        try:
            response = self.ecr_client.get_repository_policy(
```

```
                repositoryName=repository_name
            )
            return response["policyText"]
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
                    "Couldn't get repository policy for repository %s. Here's why
%s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「GetRepositoryPolicy」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_repository_name = 'my-repository'
        oo_result = lo_ecr->getrepositorypolicy(
          iv_repositoryname = iv_repository_name ).
        DATA(lv_policy_text) = oo_result->get_policytext( ).
        MESSAGE 'Repository policy retrieved.' TYPE 'I'.
      CATCH /aws1/cx_ecrrepositorynotfndex.
        MESSAGE 'Repository not found.' TYPE 'I'.
      CATCH /aws1/cx_ecrrepositoryplynot00.
```

```
        MESSAGE 'Repository policy not found.' TYPE 'I'.
    ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスの[GetRepositoryPolicy](#)」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 Amazon ECR 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 **ListImages**で を使用する

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

CLI

AWS CLI

リポジトリ内のイメージを一覧表示するには

次の list-images の例は、cluster-autoscaler リポジトリ内のイメージのリストを表示します。

```
aws ecr list-images \
    --repository-name cluster-autoscaler
```

出力:

```
{
    "imageIds": [
        {
            "imageDigest":
 "sha256:99c6fb4377e9a420a1eb3b410a951c9f464eff3b7dbc76c65e434e39b94b6570",
            "imageTag": "v1.13.8"
        },
        {
            "imageDigest":
 "sha256:99c6fb4377e9a420a1eb3b410a951c9f464eff3b7dbc76c65e434e39b94b6570",
            "imageTag": "v1.13.7"
        },
        {
```

```
            "imageDigest":
 "sha256:4a1c6567c38904384ebc64e35b7eeddd8451110c299e3368d2210066487d97e5",
            "imageTag": "v1.13.6"
        }
    ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「ListImages」を参照してください。

Rust

SDK for Rust

> **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。

```
async fn show_images(
    client: &aws_sdk_ecr::Client,
    repository: &str,
) -> Result<(), aws_sdk_ecr::Error> {
    let rsp = client
        .list_images()
        .repository_name(repository)
        .send()
        .await?;

    let images = rsp.image_ids();

    println!("found {} images", images.len());

    for image in images {
        println!(
            "image: {}:{}",
            image.image_tag().unwrap(),
            image.image_digest().unwrap()
        );
    }
```

```
    Ok(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「ListImages」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 Amazon ECR 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 **PushImageCmd**で を使用する

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

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。

```
   /**
    * Pushes a Docker image to an Amazon Elastic Container Registry (ECR)
repository.
    *
    * @param repoName  the name of the ECR repository to push the image to.
    * @param imageName the name of the Docker image.
    */
   public void pushDockerImage(String repoName, String imageName) {
       System.out.println("Pushing " + imageName + " to Amazon ECR will take a
few seconds.");
       CompletableFuture<AuthConfig> authResponseFuture =
getAsyncClient().getAuthorizationToken()
           .thenApply(response -> {
               String token =
response.authorizationData().get(0).authorizationToken();
```

```
                String decodedToken = new
 String(Base64.getDecoder().decode(token));
                String password = decodedToken.substring(4);

                DescribeRepositoriesResponse descrRepoResponse =
 getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
 descrRepoResponse.repositories().stream().filter(r ->
 r.repositoryName().equals(repoName)).findFirst().orElse(null);
                assert repoData != null;
                String registryURL = repoData.repositoryUri().split("/")[0];

                AuchConfig authConfig = new AuthConfig()
                    .withUsername("AWS")
                    .withPassword(password)
                    .withRegistryAddress(registryURL);
                return authConfig;
            })
            .thenCompose(authConfig -> {
                DescribeRepositoriesResponse descrRepoResponse =
 getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
 descrRepoResponse.repositories().stream().filter(r ->
 r.repositoryName().equals(repoName)).findFirst().orElse(null);
                getDockerClient().tagImageCmd(imageName + ":latest",
 repoData.repositoryUri() + ":latest", imageName).exec();
                try {

 getDockerClient().pushImageCmd(repoData.repositoryUri()).withTag("echo-
text").withAuthConfig(authConfig).start().awaitCompletion();
                    System.out.println("The " + imageName + " was pushed to
 ECR");

                } catch (InterruptedException e) {
                    throw (RuntimeException) e.getCause();
                }
                return CompletableFuture.completedFuture(authConfig);
            });

        authResponseFuture.join();
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「PushImageCmd」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。

```
    /**
     * Pushes a Docker image to an Amazon Elastic Container Registry (ECR)
repository.
     *
     * @param repoName the name of the ECR repository to push the image to.
     * @param imageName the name of the Docker image.
     */
    suspend fun pushDockerImage(
        repoName: String,
        imageName: String,
    ) {
        println("Pushing $imageName to $repoName will take a few seconds")
        val authConfig = getAuthConfig(repoName)

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val desRequest =
                DescribeRepositoriesRequest {
                    repositoryNames = listOf(repoName)
                }

            val describeRepoResponse = ecrClient.describeRepositories(desRequest)
            val repoData =
                describeRepoResponse.repositories?.firstOrNull
{ it.repositoryName == repoName }
                    ?: throw RuntimeException("Repository not found: $repoName")
```

```
            val tagImageCmd = getDockerClient()?.tagImageCmd("$imageName",
"${repoData.repositoryUri}", imageName)
            if (tagImageCmd != null) {
                tagImageCmd.exec()
            }
            val pushImageCmd =
                repoData.repositoryUri?.let {
                    dockerClient?.pushImageCmd(it)
                        // ?.withTag("latest")
                        ?.withAuthConfig(authConfig)
                }

            try {
                if (pushImageCmd != null) {
                    pushImageCmd.start().awaitCompletion()
                }
                println("The $imageName was pushed to Amazon ECR")
            } catch (e: IOException) {
                throw RuntimeException(e)
            }
        }
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「PushImageCmd」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 Amazon ECR 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 **PutLifeCyclePolicy**で を使用する

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

CLI

AWS CLI

ライフサイクルポリシーを作成するには

次の put-lifecycle-policy 例では、アカウントのデフォルトレジストリに指定されたリポジトリのライフサイクルポリシーを作成します。

```
aws ecr put-lifecycle-policy \
    --repository-name "project-a/amazon-ecs-sample" \
    --lifecycle-policy-text "file://policy.json"
```

policy.json の内容:

```
{
   "rules": [
       {
           "rulePriority": 1,
           "description": "Expire images older than 14 days",
           "selection": {
               "tagStatus": "untagged",
               "countType": "sinceImagePushed",
               "countUnit": "days",
               "countNumber": 14
           },
           "action": {
               "type": "expire"
           }
       }
   ]
}
```

出力:

```
{
   "registryId": "<aws_account_id>",
   "repositoryName": "project-a/amazon-ecs-sample",
   "lifecyclePolicyText": "{\"rules\":[{\"rulePriority\":1,\"description\":
\"Expire images older than 14 days\",\"selection\":{\"tagStatus\":\"untagged\",
\"countType\":\"sinceImagePushed\",\"countUnit\":\"days\",\"countNumber\":14},
\"action\":{\"type\":\"expire\"}}]}"
}
```

詳細については、「Amazon ECR ユーザーガイド」の「ライフサイクルポリシー」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「PutLifeCyclePolicy」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def put_lifecycle_policy(self, repository_name: str, lifecycle_policy_text:
 str):
        """
        Puts a lifecycle policy for an ECR repository.

        :param repository_name: The name of the repository to put the lifecycle
 policy for.
        :param lifecycle_policy_text: The lifecycle policy text to put.
        """
        try:
            self.ecr_client.put_lifecycle_policy(
                repositoryName=repository_name,
```

```
                lifecyclePolicyText=lifecycle_policy_text,
            )
            print(f"Put lifecycle policy for repository {repository_name}.")
        except ClientError as err:
            logger.error(
                "Couldn't put lifecycle policy for repository %s. Here's why %s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise
```

有効期限ポリシーを設定する例。

```
    def put_expiration_policy(self):
        """
        Puts an expiration policy on the ECR repository.
        """
        policy_json = {
            "rules": [
                {
                    "rulePriority": 1,
                    "description": "Expire images older than 14 days",
                    "selection": {
                        "tagStatus": "any",
                        "countType": "sinceImagePushed",
                        "countUnit": "days",
                        "countNumber": 14,
                    },
                    "action": {"type": "expire"},
                }
            ]
        }

        self.ecr_wrapper.put_lifecycle_policy(
            self.repository_name, json.dumps(policy_json)
        )
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「PutLifeCyclePolicy」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_repository_name = 'my-repository'
        " iv_lifecycle_policy_text = '{"rules":
[{"rulePriority":1,"description":"Expire images older than 14 days",...}]}'
        lo_ecr->putlifecyclepolicy(
          iv_repositoryname = iv_repository_name
          iv_lifecyclepolicytext = iv_lifecycle_policy_text ).
        MESSAGE |Lifecycle policy set for repository { iv_repository_name }.|
 TYPE 'I'.
      CATCH /aws1/cx_ecrrepositorynotfndex.
        MESSAGE 'Repository not found.' TYPE 'I'.
      CATCH /aws1/cx_ecrvalidationex.
        MESSAGE 'Invalid lifecycle policy format.' TYPE 'I'.
    ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスのPutLifeCyclePolicy」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 Amazon ECR 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 **SetRepositoryPolicy**で を使用する

次のサンプルコードは、SetRepositoryPolicy 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。

- 基本を学ぶ

CLI

AWS CLI

リポジトリのリポジトリポリシーを設定する場合

次の set-repository-policy の例では、ファイルに含まれるリポジトリポリシーを cluster-autoscaler リポジトリにアタッチします。

```
aws ecr set-repository-policy \
    --repository-name cluster-autoscaler \
    --policy-text file://my-policy.json
```

my-policy.json の内容:

```
{
    "Version":"2012-10-17",
    "Statement" : [
        {
            "Sid" : "allow public pull",
            "Effect" : "Allow",
            "Principal" : "*",
            "Action" : [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ]
        }
    ]
}
```

出力:

```
{
    "registryId": "012345678910",
    "repositoryName": "cluster-autoscaler",
    "policyText": "{\n  \"Version\" : \"2008-10-17\",\n  \"Statement\" :
 [ {\n    \"Sid\" : \"allow public pull\",\n    \"Effect\" : \"Allow\",\n
 \"Principal\" : \"*\",\n    \"Action\" : [ \"ecr:BatchCheckLayerAvailability\",
 \"ecr:BatchGetImage\", \"ecr:GetDownloadUrlForLayer\" ]\n  } ]\n}"
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「SetRepositoryPolicy」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。

```
   /**
    * Sets the repository policy for the specified ECR repository.
    *
    * @param repoName the name of the ECR repository.
    * @param iamRole  the IAM role to be granted access to the repository.
    * @throws RepositoryPolicyNotFoundException if the repository policy does
not exist.
    * @throws EcrException                      if there is an unexpected error
setting the repository policy.
    */
   public void setRepoPolicy(String repoName, String iamRole) {
       /*
         This example policy document grants the specified AWS principal the
permission to perform the
         `ecr:BatchGetImage` action. This policy is designed to allow the
specified principal
         to retrieve Docker images from the ECR repository.
        */
       String policyDocumentTemplate = """
           {
            "Version":"2012-10-17",
            "Statement" : [ {
              "Sid" : "new statement",
              "Effect" : "Allow",
              "Principal" : {
                "AWS" : "%s"
              },
              "Action" : "ecr:BatchGetImage"
```

```
              } ]
            }
             """;

        String policyDocument = String.format(policyDocumentTemplate, iamRole);
        SetRepositoryPolicyRequest setRepositoryPolicyRequest =
SetRepositoryPolicyRequest.builder()
            .repositoryName(repoName)
            .policyText(policyDocument)
            .build();

        CompletableFuture<SetRepositoryPolicyResponse> response =
getAsyncClient().setRepositoryPolicy(setRepositoryPolicyRequest);
        response.whenComplete((resp, ex) -> {
            if (resp != null) {
                System.out.println("Repository policy set successfully.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof RepositoryPolicyNotFoundException) {
                    throw (RepositoryPolicyNotFoundException) cause;
                } else if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
                    String errorMessage = "Unexpected error: " +
cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                }
            }
        });
        response.join();
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「SetRepositoryPolicy」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。

```
/**
 * Sets the repository policy for the specified ECR repository.
 *
 * @param repoName the name of the ECR repository.
 * @param iamRole the IAM role to be granted access to the repository.
 */
suspend fun setRepoPolicy(
    repoName: String?,
    iamRole: String?,
) {
    val policyDocumentTemplate =
        """
         {
          "Version":"2012-10-17",
          "Statement" : [ {
            "Sid" : "new statement",
            "Effect" : "Allow",
            "Principal" : {
              "AWS" : "$iamRole"
            },
            "Action" : "ecr:BatchGetImage"
          } ]
        }

        """.trimIndent()
    val setRepositoryPolicyRequest =
        SetRepositoryPolicyRequest {
            repositoryName = repoName
            policyText = policyDocumentTemplate
        }
```

```
        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val response =
 ecrClient.setRepositoryPolicy(setRepositoryPolicyRequest)
            if (response != null) {
                println("Repository policy set successfully.")
            }
        }
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「SetRepositoryPolicy」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。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def set_repository_policy(self, repository_name: str, policy_text: str):
        """
```

```
        Sets the policy for an ECR repository.

        :param repository_name: The name of the repository to set the policy for.
        :param policy_text: The policy text to set.
        """
        try:
            self.ecr_client.set_repository_policy(
                repositoryName=repository_name, policyText=policy_text
            )
            print(f"Set repository policy for repository {repository_name}.")
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
                    "Couldn't set repository policy for repository %s. Here's why
%s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise
```

IAM ロールにダウンロードアクセス権を付与する例。

```
   def grant_role_download_access(self, role_arn: str):
        """
        Grants the specified role access to download images from the ECR
repository.

        :param role_arn: The ARN of the role to grant access to.
        """
        policy_json = {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Sid": "AllowDownload",
                    "Effect": "Allow",
                    "Principal": {"AWS": role_arn},
                    "Action": ["ecr:BatchGetImage"],
```

```
                }
            ],
        }

        self.ecr_wrapper.set_repository_policy(
            self.repository_name, json.dumps(policy_json)
        )
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「SetRepositoryPolicy」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_repository_name = 'my-repository'
    " iv_policy_text = '{"Version":"2012-10-17",       "Statement":[...]}'
    lo_ecr->setrepositorypolicy(
      iv_repositoryname = iv_repository_name
      iv_policytext = iv_policy_text ).
    MESSAGE |Policy set for repository { iv_repository_name }.| TYPE 'I'.
  CATCH /aws1/cx_ecrrepositorynotfndex.
    MESSAGE 'Repository not found.' TYPE 'I'.
ENDTRY.
```

- API の詳細については、 AWS SDK for SAP ABAP API リファレンスの「SetRepositoryPolicy」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 Amazon ECR 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 **StartLifecyclePolicyPreview**で を使用する

次のサンプルコードは、StartLifecyclePolicyPreview 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。

- 基本を学ぶ

CLI

AWS CLI

ライフサイクルポリシーのプレビューを作成する場合

次の start-lifecycle-policy-preview の例では、指定されたリポジトリに対して JSON ファイルで定義されたライフサイクルポリシーのプレビューを作成します。

```
aws ecr start-lifecycle-policy-preview \
    --repository-name "project-a/amazon-ecs-sample" \
    --lifecycle-policy-text "file://policy.json"
```

policy.json の内容:

```
{
   "rules": [
       {
           "rulePriority": 1,
           "description": "Expire images older than 14 days",
           "selection": {
               "tagStatus": "untagged",
               "countType": "sinceImagePushed",
               "countUnit": "days",
               "countNumber": 14
           },
           "action": {
               "type": "expire"
```

```
            }
        }
    ]
}
```

出力:

```
{
    "registryId": "012345678910",
    "repositoryName": "project-a/amazon-ecs-sample",
    "lifecyclePolicyText": "{\n    \"rules\": [\n        {\n
 \"rulePriority\": 1,\n            \"description\": \"Expire images older than 14
 days\",\n            \"selection\": {\n                \"tagStatus\": \"untagged
\",\n                \"countType\": \"sinceImagePushed\",\n
 \"countUnit\": \"days\",\n                \"countNumber\": 14\n            },\n
           \"action\": {\n                \"type\": \"expire\"\n            }\n
      }\n    ]\n}\n",
    "status": "IN_PROGRESS"
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「StartLifecyclePolicyPreview」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。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
(Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     * @throws EcrException             if there is an error retrieving the image
information from Amazon ECR.
```

```
     * @throws CompletionException      if the asynchronous operation completes
 exceptionally.
     */
    public void verifyImage(String repositoryName, String imageTag) {
        DescribeImagesRequest request = DescribeImagesRequest.builder()
            .repositoryName(repositoryName)
            .imageIds(ImageIdentifier.builder().imageTag(imageTag).build())
            .build();

        CompletableFuture<DescribeImagesResponse> response =
 getAsyncClient().describeImages(request);
        response.whenComplete((describeImagesResponse, ex) -> {
            if (ex != null) {
                if (ex instanceof CompletionException) {
                    Throwable cause = ex.getCause();
                    if (cause instanceof EcrException) {
                        throw (EcrException) cause;
                    } else {
                        throw new RuntimeException("Unexpected error: " +
 cause.getMessage(), cause);
                    }
                } else {
                    throw new RuntimeException("Unexpected error: " +
 ex.getCause());
                }
            } else if (describeImagesResponse != null &&
!
describeImagesResponse.imageDetails().isEmpty()) {
                System.out.println("Image is present in the repository.");
            } else {
                System.out.println("Image is not present in the repository.");
            }
        });

        // Wait for the CompletableFuture to complete.
        response.join();
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「StartLifecyclePolicyPreview」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。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     */
    suspend fun verifyImage(
        repoName: String?,
        imageTagVal: String?,
    ) {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        require(!(imageTagVal == null || imageTagVal.isEmpty())) { "Image tag
 cannot be null or empty" }

        val imageId =
            ImageIdentifier {
                imageTag = imageTagVal
            }
        val request =
            DescribeImagesRequest {
                repositoryName = repoName
                imageIds = listOf(imageId)
            }

        EcrClient.fromEnvironment { region = "us-east-1" }.use { ecrClient ->
            val describeImagesResponse = ecrClient.describeImages(request)
            if (describeImagesResponse != null && !
describeImagesResponse.imageDetails?.isEmpty()!!) {
                println("Image is present in the repository.")
```

```
            } else {
                println("Image is not present in the repository.")
            }
        }
    }
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「StartLifecyclePolicyPreview」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 Amazon ECR 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。

Amazon ECR のサービスクォータ

次の表に、Amazon Elastic Container Registry (Amazon ECR) のデフォルトのサービスクォータを示します。

名前	デフォルト	引き上げ可能	説明
24 時間あたりの基本イメージスキャン	サポートされている各リージョン: 100,000	いいえ	現在のアカウントとリージョンで、基本スキャンを使用して 24 時間以内にスキャンできるイメージの最大数。この制限には、プッシュ時スキャンと手動スキャンの両方が含まれます。
レプリケーション設定のルールごとのフィルター数	サポートされている各リージョン: 100	いいえ	レプリケーション設定のルールごとのフィルターの最大数。
リポジトリあたりのイメージ数	サポートされている各リージョン: 100,000	あり	リポジトリあたりのイメージの最大数。
レイヤーパート	サポートされている各リージョン: 4,200	いいえ	レイヤーパートの最大数これは、Amazon ECR API アクションを直接使用して、イメージプッシュオペレーションのマルチパートアップロードを開始している場合にのみ適用されます。

名前	デフォルト	引き上げ可能	説明
ライフサイクルポリシーの長さ	サポートされている各リージョン: 30,720	いいえ	ライフサイクルポリシーの最大文字数。
レイヤーパートの最大サイズ	サポートされている各リージョン: 10	いいえ	レイヤーパートの最大サイズ (MiB)。これは、Amazon ECR API アクションを直接使用して、イメージプッシュオペレーションのマルチパートアップロードを開始している場合にのみ適用されます。
レイヤーの最大サイズ	サポートされている各リージョン: 52,000	いいえ	レイヤーの最大サイズ (MiB)。
レイヤーパートの最小サイズ	サポートされている各リージョン : 5	いいえ	レイヤーパートの最小サイズ (MiB)。これは、Amazon ECR API アクションを直接使用して、イメージプッシュオペレーションのマルチパートアップロードを開始している場合にのみ適用されます。

名前	デフォルト	引き上げ可能	説明
レジストリーごとのプルスルーキャッシュルール	サポートされている各リージョン: 50	いいえ	プルスルーキャッシュルールの最大数。
BatchCheckLayerAvailability リクエストのレート	サポートされている各リージョン: 1,000/秒	あり	現在のリージョンで 1 秒あたりに実行できる BatchCheckLayerAvailability リクエストの最大数。イメージがリポジトリにプッシュされると、イメージレイヤーごとに以前にアップロードされたかどうかが確認されます。アップロードされている場合、そのイメージレイヤーはスキップされます。

名前	デフォルト	引き上げ可能	説明
BatchGetImage リクエストのレート	サポートされている各リージョン: 2,000/秒	あり	現在のリージョンで 1 秒あたりに実行できる BatchGetImage リクエストの最大数。イメージがプルされると、BatchGetImage API が 1 回呼び出され、イメージマニフェストが取得されます。この API のクォータの引き上げをリクエストする場合、GetDownloadUrlForLayer の使用状況も確認してください。
CompleteLayerUpload リクエストのレート	サポートされている各リージョン: 100/秒	あり	現在のリージョンで 1 秒あたりに実行できる CompleteLayerUpload リクエストの最大数。イメージがプッシュされると、新しいイメージレイヤーごとに CompleteLayerUpload API が 1 回呼び出されて、アップロードが完了したことが確認されます。

名前	デフォルト	引き上げ可能	説明
GetAuthorizationToken リクエストのレート	サポートされている各リージョン: 500/秒	あり	現在のリージョンで 1 秒あたりに実行できる GetAuthorizationToken リクエストの最大数。
GetDownloadUrlForLayer リクエストのレート	サポートされている各リージョン: 3,000/秒	あり	現在のリージョンで 1 秒あたりに実行できる GetDownloadUrlForLayer リクエストの最大数。イメージがプルされると、まだキャッシュされていないイメージレイヤーごとに GetDownloadUrlForLayer API が 1 回呼び出されます。この API のクォータの引き上げをリクエストする場合、BatchGetImage の使用状況も確認してください。

名前	デフォルト	引き上げ可能	説明
InitiateLayerUpload リクエストのレート	サポートされている各リージョン: 100/秒	あり	現在のリージョンで 1 秒あたりに実行できる InitiateLayerUpload リクエストの最大数。イメージがプッシュされると、まだアップロードされていないイメージレイヤーごとに InitiateLayerUpload API が 1 回呼び出されます。イメージレイヤーがアップロードされたかどうかは、BatchCheckLayerAvailability API アクションによって決定されます。
PutImage リクエストのレート	サポートされている各リージョン: 10/秒	あり	現在のリージョンで 1 秒あたりに実行できる PutImage リクエストの最大数。イメージがプッシュされ、すべての新しいイメージレイヤーがアップロードされると、PutImage API が 1 回呼び出され、イメージマニフェスト、およびイメージに関連付けられたタグが作成または更新されます。

名前	デフォルト	引き上げ可能	説明
UploadLayerPart リクエストのレート	サポートされている各リージョン: 500/秒	あり	現在のリージョンで 1 秒あたりに実行できる UploadLayerPart リクエストの最大数。イメージがプッシュされると、新しい各イメージレイヤーがいくつかに分けてアップロードされ、UploadLayerPart API が新しいイメージレイヤーパートごとに 1 回呼び出されます。
イメージスキャンのレート	サポートされている各リージョン: 1	[いいえ]	24 時間あたりのイメージスキャンの最大数。
登録済みリポジトリ	サポートされている各リージョン: 100,000	あり	このアカウントで現在のリージョンに作成できるリポジトリの最大数。
ライフサイクルポリシーあたりのルール	サポートされている各リージョン: 50	いいえ	ライフサイクルポリシーのルールの最大数
レプリケーション設定ごとのルール	サポートされている各リージョン: 10	いいえ	レプリケーション設定のルールの最大数。

名前	デフォルト	引き上げ可能	説明
イメージあたりのタグ	サポートされている各リージョン: 1,000	いいえ	イメージあたりのタグの最大数。
レプリケーション設定のすべてのルール全体における一意のレプリケート先	サポートされている各リージョン: 25	いいえ	レプリケーション設定のすべてのルール全体における一意のレプリケート先の最大数。

での Amazon ECR サービスクォータの管理 AWS マネジメントコンソール

Amazon ECR は Service Quotas と統合されています。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 を使用すると、すべての Amazon ECR サービスクォータの値を簡単に検索できます。

Amazon ECR のサービスクォータを表示するには (AWS マネジメントコンソール)

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] リストから [Amazon Elastic Container Registry (Amazon ECR)] を選択します。

   サービスクォータリストでは、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、およびクォータ値が調整可能かどうかを確認できます。
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。

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。

API 使用量メトリクスをモニタリングするための CloudWatch アラームの作成

Amazon ECR は、レジストリ認証、イメージプッシュ、イメージプルアクションに関連する各 APIs AWS のサービスクォータに対応する CloudWatch 使用状況メトリクスを提供します。Service Quotas コンソールでは、使用状況をグラフで可視化し、使用量がサービスクォータに近づくと警告するアラームを設定できます。詳細については、「Amazon ECR 使用状況メトリクス」を参照してください。

以下の手順を使用して、Amazon ECR API 使用量メトリクスのいずれかに基づいて CloudWatch アラームを作成します。

Amazon ECR 使用量クォータに基づいてアラームを作成するには (AWS マネジメントコンソール)

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] リストから [Amazon Elastic Container Registry (Amazon ECR)] を選択します。
4. [Service quotas (サービスクォータ)] リストで、アラームを作成する Amazon ECR 使用量クォータを選択します。
5. Amazon CloudWatch Events アラームのセクションで [Create] (作成) を選択します。
6. [アラームのしきい値] で、適用されたクォータ値からアラーム値として設定する値の割合を選択します。
7. [アラーム名] にアラームの名前を入力し、[Create (作成)] を選択します。

Amazon ECR のトラブルシューティング

この章では、Amazon ECR の診断情報を見つけるのに役立つ情報と、一般的な問題やエラーメッセージに対するトラブルシューティングの手順を提供します。

トピック

- Amazon ECR の使用時の Docker コマンドおよび問題のトラブルシューティング
- Amazon ECR エラーメッセージのトラブルシューティング

Amazon ECR の使用時の Docker コマンドおよび問題のトラブルシューティング

場合によっては、Amazon ECR に対して Docker コマンドを実行すると、エラーメッセージが表示されることがあります。一般的なエラーメッセージと考えられる解決策を以下に説明します。

トピック

- Docker ログに予想されるエラーメッセージが含まれていない
- Amazon ECR レポジトリからイメージをプルするときのエラー: "ファイルシステムの検証に失敗しました" または "404: イメージが見つかりません"
- Amazon ECR からイメージをプルする際のエラー: "ファイルシステムレイヤーの検証に失敗しました"
- レポジトリへのプッシュの際の HTTP 403 エラー、または "基本的な認証情報がありません" エラー

Docker ログに予想されるエラーメッセージが含まれていない

Docker 関連の問題のデバッグを開始するには、最初にホストインスタンスで実行している Docker デーモンで Docker デバッグ出力をオンにします。Amazon ECS コンテナインスタンスで Amazon ECR からプルされたイメージを使用している場合は、「Amazon Elastic Container Service デベロッパーガイド」の「Docker デーモンからの詳細な出力の設定」を参照してください。

Amazon ECR レポジトリからイメージをプルするときのエラー: "ファイルシステムの検証に失敗しました" または "404: イメージが見つかりません"

docker pull コマンドを使用すると、Docker 1.9 以降で Amazon ECR レポジトリからイメージをプルするときにエラー `Filesystem verification failed` が表示されます。Docker 1.9 より前のバージョンを使用するときはエラー `404: Image not found` が表示される場合があります。

考えられる理由とその説明を以下に示します。

ローカルディスクがいっぱいである

docker pull を実行しているローカルディスクがいっぱいである場合、ローカルファイルで計算された SHA-1 ハッシュは、Amazon ECR で計算されたものとは異なる可能性があります。ローカルディスクに、プルしている Docker イメージを保存するのに十分な空き容量があることを確認します。新しいイメージの容量を確保するために、古いイメージを削除することもできます。docker images コマンドを使用して、ローカルにダウンロードしたすべての Docker イメージのリストとそのサイズを表示します。

ネットワークエラーにより、クライアントがリモートリポジトリに接続できない

Amazon ECR レポジトリへの呼び出しでは、インターネットへの機能している接続が必要です。ネットワーク設定を確認し、他のツールやアプリケーションがインターネット上のリソースにアクセスできることを確認します。プライベートサブネットの Amazon EC2 インスタンスで docker pull を実行している場合は、そのサブネットにインターネットへのルートがあることを確認します。ネットワークアドレス変換 (NAT) サーバーまたはマネージド NAT ゲートウェイを使用します。

現時点では、Amazon ECR リポジトリへの呼び出しでは、会社のファイアウォールから Amazon Simple Storage Service (Amazon S3) へのネットワークアクセスも必要です。組織で、サービスエンドポイントを許可するファイアウォールソフトウェアまたは NAT デバイスを使用している場合、現在のリージョンの Amazon S3 サービスエンドポイントが許可されていることを確認します。

HTTP プロキシの背後で Docker を使用している場合は、適切なプロキシ設定を使用して Docker を設定できます。詳細については、Docker ドキュメントの「HTTP プロキシ」を参照してください。

Amazon ECR からイメージをプルする際のエラー: "ファイルシステムレイヤーの検証に失敗しました"

docker pull コマンドを使用してイメージをプルするときに、エラー `image image-name not found` が表示される場合があります。Docker のログを調べると、次のようなエラーが見つかる場合があります。

```
filesystem layer verification failed for digest sha256:2b96f...
```

このエラーは、イメージの 1 つ以上のレイヤーがダウンロードに失敗したことを示します。考えられる理由とその説明を以下に示します。

古いバージョンの Docker 使用している

> このエラーは、Docker 1.10 より前のバージョンを使用している場合に、まれに発生することがあります。Docker クライアントを 1.10 以降にアップグレードします。

クライアントでネットワークエラーまたはディスクエラーが発生した

> `Filesystem verification failed` メッセージについて前に説明したように、ディスクがいっぱいであるか、ネットワークの問題により、1 つ以上のレイヤーをダウンロードできない可能性があります。上記の推奨事項に従って、ファイルシステムがいっぱいでないこと、およびネットワーク内から Amazon S3 へのアクセスを有効にしたことを確認します。

レポジトリへのプッシュの際の HTTP 403 エラー、または "基本的な認証情報がありません" エラー

aws ecr get-login-password コマンドを使用して Docker に対して正常に認証されても、`HTTP 403 (Forbidden)` エラーが発生したり、docker push コマンドまたは docker pull コマンドからのエラーメッセージ `no basic auth credentials` が表示されたりする場合があります。この問題の既知の原因をいくつか次に示します。

別のリージョンに対して認証されている

> 認証リクエストは特定のリージョンに関連付けられていて、リージョン間では使用できません。たとえば、米国西部 (オレゴン)リージョン から認可トークンを取得する場合、これを使用して、米国東部 (バージニア北部) リージョンのリポジトリに対して認証を行うことはできません。この問題を解決するには、リポジトリが存在する同じリージョンから認証トークンを取得したことを

確認してください。詳細については、「the section called “レジストリの認証”」を参照してください。

プッシュ先として認証したリポジトリに対するアクセス許可がない

プッシュ先のリポジトリに対するアクセス許可がありません。詳細については、「Amazon ECR でのプライベートリポジトリポリシー」を参照してください。

トークンの有効期限が切れた。

GetAuthorizationToken オペレーションを使用して取得した認可トークンのデフォルトの有効期限は 12 時間です。

wincred 認証情報マネージャーのバグ

Windows 用 Docker の一部のバージョンでは、wincred という認証情報マネージャーを使用しています。この認証情報マネージャーでは、aws ecr get-login-password によって生成された Docker ログインコマンドが適切に処理されません (詳細については、「CredsStore fails with private repositories」を参照してください)。出力される Docker ログインコマンドは実行できますが、イメージをプッシュまたはプルしようとすると、コマンドは失敗します。これに対処するには、aws ecr get-login-password から出力される Docker ログインコマンドのレジストリ引数から https:// スキームを削除します。HTTPS スキームのない Docker ログインコマンドの例を次に示します。

```
docker login -u AWS -p <password> <aws_account_id>.dkr.ecr.<region>.amazonaws.com
```

Amazon ECR エラーメッセージのトラブルシューティング

Amazon ECR コンソールまたは AWS CLI を通じて開始された API コールがエラーメッセージで終了することがあります。一般的なエラーメッセージと考えられる解決策を以下に説明します。

HTTP 429: リクエストが多すぎる、または ThrottleException

1 つ以上の Amazon ECR アクションまたは API コールから、429: Too Many Requests エラーまたは ThrottleException エラーを受け取ることがあります。これは、短い間隔で Amazon ECR の単一のエンドポイントを繰り返し呼び出していて、リクエストがスロットリングされていることを示します。スロットリングは、1 人のユーザーから単一のエンドポイントへの呼び出しが、期間中に特定のしきい値を超えたときに発生します。

Amazon ECR での各 API オペレーションには、レートスロットルが関連付けられています。たとえば、GetAuthorizationToken アクションのスロットリングは 20 TPS (トランザクショ

ン/秒) で、最大 200 TPS のバーストが可能です。各リージョンで、各アカウントには、最大 200 GetAuthorizationToken クレジットを保存できるバケットが割り当てられ、1 秒あたり 20 クレジットの割合で補充されます。バケットに 200 クレジットがある場合、1 秒あたり 200 GetAuthorizationToken API トランザクションに達すると、1 秒あたり 20 トランザクションが無期限に維持されます。Amazon ECR API のレート制限の詳細については、「Amazon ECR のサービスクォータ」を参照してください。

スロットリングエラーを処理するには、増分バックオフをコードに含めて再試行関数を実装します。詳細については、「AWS SDK とツールのリファレンスガイド」の「再試行動作」を参照してください。もう 1 つのオプションは、レート制限の引き上げをリクエストすることです。これを行うには、Service Quotas コンソールを使用します。詳細については、「での Amazon ECR サービスクォータの管理 AWS マネジメントコンソール」を参照してください。

HTTP 403: "ユーザー [arn] は [operation] の実行を許可されていません"

Amazon ECR を使用してアクションを実行しようとすると、次のエラーを受け取ることがあります。

```
$ aws ecr get-login-password
A client error (AccessDeniedException) occurred when calling the GetAuthorizationToken
 operation:
    User: arn:aws:iam::account-number:user/username is not authorized to perform:
    ecr:GetAuthorizationToken on resource: *
```

これは、Amazon ECR を使用するアクセス権限がユーザーに付与されていないか、それらのアクセス権限が正しく設定されていないことを示します。特に、Amazon ECR レポジトリに対してアクションを実行している場合は、そのリポジトリにアクセスする権限がユーザーに付与されていることを確認します。Amazon ECR の許可の作成と検証の詳細については、「Amazon Elastic Container Registry の Identity and Access Management」を参照してください。

HTTP 404: "レポジトリは存在しません" エラー

現在存在しない Docker Hub リポジトリを指定すると、Docker Hub はそのレポジトリを自動的に作成します。Amazon ECR では、レポジトリを使用するには、その前に新しいリポジトリを明示的に作成する必要があります。これにより、(タイプミスなどの原因で) 新しいレポジトリが間違って作成されるのを防止し、適切なセキュリティアクセスポリシーが明示的に新しいレポジトリに割り当てられます。レポジトリの作成方法の詳細については、「Amazon ECR プライベートリポジトリ」を参照してください。

エラー: Cannot perform an interactive login from a non TTY device (TTY 以外のデバイスから対話型ログインを実行できません)

`Cannot perform an interactive login from a non TTY device` エラーが表示される場合は、以下のトラブルシューティング手順を試してください。

- AWS CLI バージョン 2 を使用し、システムでバージョン 1 AWS CLI の競合バージョンがないことを確認します。詳細については、「Installing or updating the latest version of the AWS CLI」を参照してください。
- が有効な認証情報 AWS CLI で設定されていることを確認します。詳細については、「Installing or updating the latest version of the AWS CLI」を参照してください。
- AWS CLI コマンドの構文が正しいことを確認します。

Amazon ECR での Podman の使用

Podman と Amazon ECR を併用することで、組織は Podman のセキュリティとシンプルさを活用しながら、コンテナイメージ管理における Amazon ECR のスケーラビリティと信頼性を享受できます。概説されている手順とコマンドに従うことで、開発者と管理者はコンテナワークフローを合理化し、セキュリティを強化し、リソース使用率を最適化できます。コンテナ化が勢いを増し続ける中、Podman と Amazon ECR を使用することで、コンテナ化されたアプリケーションを管理およびデプロイするための堅牢で柔軟なソリューションが提供されます。

Amazon ECR での認証に対する Podman の使用

Podman を使用して Amazon ECR を操作する前に、認証が必要です。これを行うには、`aws ecr get-login-password` コマンドを実行して認証トークンを取得し、そのトークンを `podman login` コマンドで使用して Amazon ECR で認証します。

```
aws ecr get-login-password --region <region> | podman login --username AWS --password-stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com
```

Podman とともに Amazon ECR 認証情報ヘルパーを使用する

Amazon ECR には、Podman で動作する Docker 認証情報ヘルパーが用意されています。認証情報ヘルパーを使用すると、Amazon ECR に対してイメージをプッシュおよびプルするときに、Docker 認証情報をより簡単に保存および使用できます。インストールおよび設定手順については、「Amazon ECR Docker 認証情報ヘルパー」を参照してください。

> **Important**
>
> Podman は docker-creds-helper 仕様を部分的にしかサポートしていません。Podman は Docker 設定で `credHelpers` キーワードをサポートしていますが、`credsStore` キーワードはサポートしていません。
> Podman で Amazon ECR 認証情報ヘルパーを使用するには、Docker 設定ファイルを次の `credHelpers` 形式で設定します。
>
> ```
> {
>     "credHelpers": {
>         "public.ecr.aws": "ecr-login",
> ```

```
        "<aws_account_id>.dkr.ecr.<region>.amazonaws.com": "ecr-login"
    }
}
```

Podman では、次の `credsStore` 設定はサポートされていません。

```
{
    "credsStore": "ecr-login"
}
```

> **Note**
>
> Amazon ECR Docker 認証情報ヘルパーは、現在、多要素認証 (MFA) をサポートしていません。

Podman を使用した Amazon ECR からのイメージのプル

認証に成功すると、完全な Amazon ECR リポジトリ URI を指定した `podman pull` コマンドを使用して、Amazon ECR からコンテナイメージをプルできます。

```
podman pull aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
```

Podman を使用した Amazon ECR のコンテナの実行

目的のイメージがプルされると、`podman run` コマンドを使用してコンテナをインスタンス化できます。

```
podman run -d aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
```

Podman を使用した Amazon ECR へのイメージのプッシュ

ローカルイメージを Amazon ECR にプッシュするには、まず `podman tag` を使用してイメージに Amazon ECR リポジトリ URI でタグ付けする必要があります。その後、`podman push` コマンドを使用してイメージを Amazon ECR にアップロードできます。

```
podman
 tag local_image:tag aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
podman push aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
```

ドキュメント履歴

次の表に、Amazon ECR の前回のリリース以後に行われたドキュメントの重要な変更を示します。また、お客様からいただいたフィードバックに対応するために、ドキュメントを頻繁に更新しています。

変更	説明	日付
Clair ベースのスキャンの廃止が完了しました	2026 年 2 月 2 日現在、Clair ベースのイメージスキャンは廃止され、すべての ECR アカウントは AWS ネイティブ基本スキャンに移行されています。Clair 固有のコンテンツを削除し、現在は AWS ネイティブが基本スキャンの唯一の実装であることを反映するようにドキュメントが更新されました。	2026 年 2 月 2 日
ECR マネージド署名	Amazon ECR は、セキュリティ体制を強化し、署名の設定に伴う運用上のオーバーヘッドを排除するために、マネージドコンテナイメージ署名をサポートするようになりました。コンテナイメージ署名を使用すると、イメージが信頼できるソースからのものであることを確認できます。詳細については、「マネージド署名」を参照してください。	2025 年 11 月 21 日
AWS PrivateLink (VPC エンドポイント) の IPv6 サポート	AWS PrivateLink を搭載した Amazon ECR VPC エンドポイントのデュアルスタック (IPv4 および IPv6) 接続のサポートが追加されました。IPv4 プライベート IP アドレスと IPv6 プライベート IP アドレスの両方でトラフィックを処理するデュアルスタック VPC エンドポイントを作成できるようになりました。詳細については、「Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。	2025 年 11 月 21 日
ECR アーカイブ機能	Amazon ECR では、長期保存のためにイメージをアーカイブするサポートが追加されました。詳細については、「Amazon ECR でのイメージのアーカイブ」を参照してください。	2025 年 11 月 19 日

変更	説明	日付
イメージタグのイミュータビリティ除外パターンのサポートを含むように更新	Amazon ECR は、リポジトリの作成と更新中にイメージタグのイミュータビリティ除外パターンを含めるようにイメージタグ付け機能を更新しました。他のすべてのタグのイミュータビリティを維持しながら、特定のタグをイミュータビリティルールから除外するワイルドカードパターン (`latest"`、`v*.beta`、`dev-*` など) を定義することで、リポジトリでタグのイミュータビリティが有効になっている場合でも更新できるタグを指定できるようになりました。詳細については、「イメージを保存するための Amazon ECR プライベートリポジトリの作成」を参照してください。	2025 年 7 月 23 日
拡張イメージスキャンを更新し、イメージ使用状況に関するインサイトを提供	Amazon ECR は、拡張イメージスキャン機能を更新し、Amazon EKS および Amazon ECS でのイメージの使用方法の可視性を追加しました。詳細については、「Amazon ECR で OS とプログラミング言語のパッケージの脆弱性を調べるためのイメージのスキャン」を参照してください。	2025 年 6 月 16 日
IPv6 サポート	IPv4 のみのエンドポイントとデュアルスタック (IPv4 および IPv6) エンドポイントの両方を使用して Amazon ECR レジストリにリクエストを行うサポートが追加されました。詳細については、「Amazon ECR レジストリにリクエストを実行する」を参照してください。	2025 年 4 月 30 日
キャッシュをプルスルーするための Amazon ECR プライベートレジストリのサポートを追加	Amazon ECR で、Amaon ECRプライベートレジストリのプルスルーキャッシュルールを作成するサポートが追加されました。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリの同期」および「プルスルーキャッシュの Amazon ECR サービスにリンクされたロール」を参照してください。	2025 年 3 月 12 日

変更	説明	日付
レジストリポリシースコープの設定のサポートを追加	Amazon ECR は、プライベートレジストリのレジストリポリシースコープを設定するサポートを追加しました。詳細については、「Private registry permissions in Amazon ECR」および「Amazon ECR private registry」を参照してください。	2024 年 12 月 23 日
AmazonEC2ContainerRegistryPullOnly – 新しいポリシー	Amazon ECR に、プル専用アクセス許可を Amazon ECR に付与する新しいポリシーが追加されました。	2024 年 10 月 10 日
CloudTrail イベントにおける Docker/OCI クライアントプロキシオペレーションのポイント先が `ecr.amazonaws.com` に変更	Docker/OCI クライアントエンドポイントに関連付けられた CloudTrail イベントにおいて、ユーザーエージェント (`userAgent`) フィールドとソース IP アドレス (`sourceIPAddress`) フィールドの値 `AWS Internal` が `ecr.amazonaws.com` に置き換えられます。例については、「例: イメージプルアクション」および「例: イメージプッシュアクション」を参照してください。	2024 年 7 月 1 日
リポジトリ作成テンプレートの新しい Amazon ECR サービスリンクロールの説明の追加。	Amazon ECR は、AWSServiceRoleForECRTemplate という名前のサービスにリンクされたロールを使用します。これにより、Amazon ECR がユーザーに代わってアクションを実行し、リポジトリ作成テンプレートアクションを実行するアクセス許可が付与されます。詳細については、「リポジトリ作成テンプレート用の Amazon ECR サービスリンクロール」を参照してください。	2024 年 6 月 20 日
`ECRTemplateServiceRolePolicy` サービスリンクロールが追加されました。	`ECRTemplateServiceRolePolicy` サービスリンクロールが追加されました。詳細については、`ECRTemplateServiceRolePolicy` を参照してください。	2024 年 6 月 20 日

変更	説明	日付
中国リージョンにクロスリージョンレプリケーションとクロスアカウントレプリケーションが追加されました。	Amazon ECR で、中国リージョンにおいて、レプリケートされるリポジトリのフィルタリングに対するサポートが追加されました。詳細については、Amazon ECR でのプライベートイメージレプリケーションを参照してください。	2024 年 5 月 15 日
プルスルーキャッシュルールへの GitLab コンテナレジストリの追加	Amazon ECR で、GitLab コンテナレジストリのプルスルーキャッシュルールの作成に対するサポートが追加されました。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリの同期」を参照してください。	2024 年 5 月 8 日
Amazon ECR ライフサイクルポリシーが更新され、ワイルドカードの使用がサポートされるようになりました	Amazon ECR は、ライフサイクルポリシールールで `tagPatternList` パラメータを使用することにより、ライフサイクルポリシーでのワイルドカードをサポートできるようになりました。詳細については、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。	2023 年 12 月 18 日
Amazon ECR リポジトリ作成テンプレート	Amazon ECR にリポジトリの作成テンプレートのサポートが追加されました。詳細については、「プルスルーキャッシュ、プッシュ時作成、またはレプリケーションアクション中に作成されたリポジトリを制御するテンプレート」を参照してください。	2023 年 11 月 15 日
Amazon ECR プルスルーキャッシュで、認証済みのアップストリームレジストリでサポートされるようになりました	Amazon ECR で、プルスルーキャッシュルールに認証が必要なアップストリームレジストリの使用がサポートされるようになりました。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリの同期」を参照してください。	2023 年 11 月 15 日

変更	説明	日付
AWSECRPullThroughCache_ServiceRolePolicy – 既存のポリシーの更新	Amazon ECR は、新しいアクセス許可を AWSECRPullThroughCache_ServiceRolePolicy ポリシーに追加しました。これらの権限により、Amazon ECR は Secrets Manager シークレットの暗号化されたコンテンツを取得できます。これは、プルスルーキャッシュルールを使用して、認証が必要なアップストリームレジストリからイメージをキャッシュする場合に必要です。	2023 年 11 月 15 日
Amazon ECR イメージ署名	Amazon ECR と に、Notary クライアントを使用したコンテナイメージ署名の作成とプッシュのサポート AWS Signer が追加されました。詳細については、「Amazon ECR でイメージに署名する」を参照してください。	2023 年 6 月 6 日
キャッシュルールのプルスルーに Kubernetes コンテナレジストリが追加	Amazon ECR で、Kubernetes コンテナレジストリのプルスルーキャッシュルールを作成するサポートが追加されました。詳細については、「アップストリームレジストリと Amazon ECR プライベートレジストリの同期」を参照してください。	2023 年 6 月 1 日
Amazon ECR 拡張スキャン期間のサポート	Amazon Inspector で、拡張スキャンが有効になっている場合にリポジトリが監視される期間を設定できるようになりました。詳細については、「Amazon Inspector でのイメージの拡張スキャン期間の変更」を参照してください。	2022 年 6 月 28 日
Amazon ECR はリポジトリのプルカウントメトリクスを Amazon CloudWatch に送信します	Amazon ECR はリポジトリのプルカウントメトリクスを Amazon CloudWatch に送信します。詳細については、「Amazon ECR リポジトリメトリクス」を参照してください。	2022 年 1 月 6 日

変更	説明	日付
拡張されたレプリケーションサポート	Amazon ECR が、レプリケートされるリポジトリのフィルタリングに対するサポートを追加しました。詳細については、「Amazon ECR でのプライベートイメージレプリケーション」を参照してください。	2021 年 9 月 21 日
AWS Amazon ECR の マネージドポリシー	Amazon ECR は、 AWS 管理ポリシーのドキュメントを追加しました。詳細については、「AWS Amazon Elastic Container Registry の マネージドポリシー」を参照してください。	2021 年 6 月 24 日
クロスリージョンおよびクロスアカウントレプリケーション	Amazon ECR で、プライベートレジストリのレプリケーション設定を構成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 ECR でのプライベートレジストリ設定」を参照してください。	2020 年 12 月 8 日
OCI アーティファクトサポート	Amazon ECR では、Open Container Initiative (OCI) アーティファクトのプッシュおよびプルがサポートされました。アーティファクトのタイプを示すために `DescribeImages` API レスポンスに新しいパラメータ `artifactMediaType` が追加されました 詳細については、「Amazon ECR プライベートリポジトリへの Helm チャートのプッシュ」を参照してください。	2020 年 8 月 24 日
保管中の暗号化	Amazon ECR で、 AWS Key Management Service (AWS KMS) に格納されたカスタマーマネージドキーでサーバー側の暗号化を使用してレポジトリを暗号化するための設定のサポートが追加されました。 詳細については、「保管中の暗号化」を参照してください。	2020 年 7 月 29 日

変更	説明	日付
マルチアーキテクチャイメージ	Amazon ECR で、マルチアーキテクチャイメージに使用される Docker マニフェストリストの作成とプッシュのサポートが追加されました。 詳細については、「Amazon ECR プライベートリポジトリへのマルチアーキテクチャイメージのプッシュ」を参照してください。	2020 年 4 月 28 日
Amazon ECR 使用状況メトリクス	Amazon ECR で、アカウントのリソースの使用状況を可視化する CloudWatch 使用量メトリクスが追加されました。また、CloudWatch と Service Quotas の両方のコンソールから CloudWatch アラームを作成し、適用されたサービスクォータに使用量が近づいたときにアラートを受けられるようになりました。 詳細については、「Amazon ECR 使用状況メトリクス」を参照してください。	2020 年 2 月 28 日
Amazon ECR サービスクォータの更新	Amazon ECR サービスクォータが更新されて、API 別のクォータを含めることができるようになりました。 詳細については、「Amazon ECR のサービスクォータ」を参照してください。	2020 年 2 月 19 日
`get-login-password` コマンドの追加	get-login-password のサポートが追加されて、シンプルで安全な方法で認可トークンを取得できるようになりました。 詳細については、「認可トークンを使用する」を参照してください。	2020 年 2 月 4 日

変更	説明	日付
イメージスキャン	コンテナイメージ内のソフトウェアの脆弱性を識別するのに役立つイメージスキャンのサポートが追加されました。Amazon ECR は、オープンソースの CoreOS Clair プロジェクトの共通脆弱性識別子 (CVE) データベースを使用し、スキャン結果のリストを提供します。 詳細については、「Amazon ECR でイメージをスキャンしてソフトウェア脆弱性がないか調べる」を参照してください。	2019 年 10 月 24 日
VPC エンドポイントポリシー	Amazon ECR インターフェイス VPC エンドポイントで IAM ポリシーを設定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 ECR VPC エンドポイントのエンドポイントポリシーを作成する」を参照してください。	2019 年 9 月 26 日
イメージタグの変更可能性	イメージタグが上書きされるのを防止するためにリポジトリの設定で変更不可のサポートが追加されました。 詳細については、「Amazon ECR でのイメージタグの上書きの防止」を参照してください。	2019 年 7 月 25 日
インターフェイス VPC エンドポイント (AWS PrivateLink)	AWS PrivateLink を使用したインターフェイス VPC エンドポイントの設定のサポートが追加されました。これにより、インターネット、NAT インスタンス、VPN 接続、または Direct Connectを経由せずに、VPC と Amazon ECR をプライベートに接続できます。 詳細については、「Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。	2019 年 1 月 25 日

変更	説明	日付
リソースのタグ付け	Amazon ECR で、メタデータタグをリポジトリに追加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 ECR でのプライベートリポジトリのタグ付け」を参照してください。	2018 年 12 月 18 日
Amazon ECR の名称変更	Amazon Elastic Container Registry の名称が変更されました (以前の名称は Amazon EC2 Container Registry)。	2017 年 11 月 21 日
ライフサイクルポリシー	Amazon ECR ライフサイクルポリシーを使用して、リポジトリ内のイメージのライフサイクル管理を指定することができます。 詳細については、「Amazon ECR でのライフサイクルポリシーを使用したイメージのクリーンアップの自動化」を参照してください。	2017 年 10 月 11 日
Amazon ECR での Docker Image Manifest 2、Schema 2 のサポート	Amazon ECR が Docker Image Manifest V2 Schema 2 (Docker バージョン 1.10 以降で使用) をサポートするようになりました。 詳細については、「Amazon ECR でのコンテナイメージマニフェスト形式のサポート」を参照してください。	2017 年 1 月 27 日
Amazon ECR の一般提供	Amazon Elastic Container Registry (Amazon ECR) は、安全でスケーラブル、信頼性の高いマネージド Docker AWS レジストリサービスです。	2015 年 12 月 21 日

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