aws

開発者ガイド

AWS HealthImaging

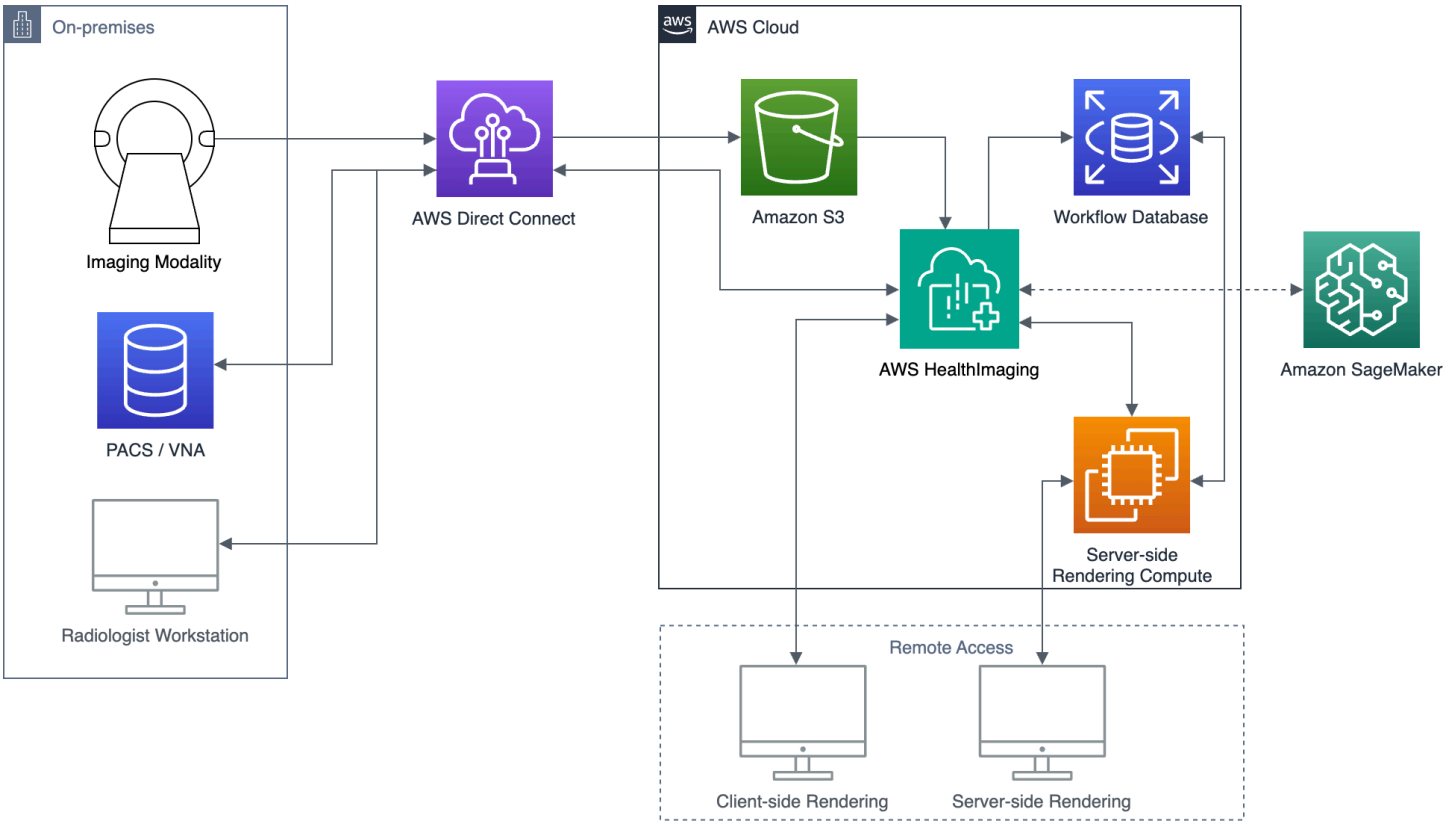
AWS HealthImaging: 開発者ガイド

Table of Contents

AWS とは HealthImaging

AWS HealthImaging は、医療プロバイダー、ライフサイエンス組織、ソフトウェアパートナーがペタバイト規模で医療イメージをクラウドに保存、分析、共有できるようにする HIPAA 認定サービスです。 HealthImaging ユースケースには以下が含まれます。

- エンタープライズイメージング – 低レイテンシーのパフォーマンスと高可用性を維持しながら、医療画像データを AWS クラウドから直接保存およびストリーミングします。
- 長期画像アーカイブ — 1 秒未満の画像取り出しアクセスを維持しながら、長期画像アーカイブのコストを削減します。
- AI/ML 開発 — 他のツールやサービスのサポートにより、画像アーカイブに対して人工知能と機械学習 (AI/ML) 推論を実行します。
- マルチモーダル分析 — 臨床画像データを AWS HealthLake （ ヘルスデータ) および AWS HealthOmics （ オミクスデータ) と組み合わせて、精度向上のためのインサイトを提供します。



AWS HealthImaging は、クラウド上に構築された医療画像アプリケーションが以前はオンプレミスでのみ可能なパフォーマンスを実現できるように、画像データ (X-Ray、CT、 MRI、Ultrasound な

ど) へのアクセスを提供します。を使用すると HealthImaging、 内の各医療画像の信頼できる単一のコピーから医療画像アプリケーションを大規模に実行することで、インフラストラクチャコストを削減できます AWS クラウド。

トピック

重要な注意点

AWS HealthImaging は、専門家による医療上の助言、診断、または治療の代用品ではなく、疾患や病状の修復、治療、緩和、予防、または診断を目的としていません。ユーザーは、臨床上の意思決定を通知することを目的としたサードパーティー製品に関連するものを含め HealthImaging、AWS の使用の一環として人間によるレビューを代入する責任があります。AWS は、適切な医療判断を適用した訓練を受けた医療専門家によるレビュー後に、患者ケアまたは臨床シナリオでのみ使用 HealthImaging してください。

AWS の機能 HealthImaging

AWS HealthImaging には以下の機能があります。

開発者に使いやすい DICOM メタデータ

AWS は、開発者にわかりやすい形式で DICOM メタデータを返すことで、アプリケーション開発 HealthImaging を簡素化します。画像データをインポートした後は、馴染みのないグループ/要素の 16 進数ではなく、わかりやすいキーワードを使用して個々のメタデータ属性にアクセスできます。患者、治験、シリーズレベルのDICOM要素は正規化されるため、アプリケーション開発者はSOPインスタンス間の不一致に対処する必要がありません。さらに、ネイティブのランタイムタイプではメタデータ属性値に直接アクセスできます。

SIMD アクセラレーションによる画像デコーディング

AWS は、高度な画像圧縮コーデックであるハイスループット JPEG 2000 (HTJ2K) でエンコードされた画像フレーム (ピクセルデータ) HealthImaging を返します。HTJ2K は、最新のプロセッサ上の単一命令複数データ (SIMD) を活用して、新しいレベルのパフォーマンスを実現します。HTJ2K は JPEG2000 よりも桁違いに高速で、他のすべての DICOM 転送構文よりも少なくとも 2 倍高速です。WASM-SIMD を利用すれば、この極めて高速なウェブビューアのフットプリントをゼロにすることができます。

ピクセルデータ検証

AWS HealthImaging は、インポート中にすべてのイメージの可逆エンコードとデコード状態をチェックすることで、組み込みのピクセルデータ検証を提供します。詳細については、「ピクセルデータ検証」を参照してください。

業界トップクラスのパフォーマンス

AWS は、効率的なメタデータエンコーディング、可逆圧縮、プログレッシブ解像度のデータアクセスにより、イメージのロードパフォーマンスの新しい標準 HealthImaging を設定します。効率的なメタデータエンコーディングにより、画像閲覧者と AI アルゴリズムは、画像データをロードしなくても DICOM 調査の内容を理解できます。高度な画像圧縮により、画質を損なうことなく画像の読み込みが速くなります。プログレッシブ解像度により、サムネイル、対象領域、低解像度のモバイルデバイスの画像読み込みがさらに速くなります。

スケーラブルな DICOM インポート

AWS HealthImaging インポートは、最新のクラウドネイティブテクノロジーを活用して、複数の DICOM 研究を並行してインポートします。履歴アーカイブは、新しいデータを求める臨床ワークロードに影響を与えることなく、迅速にインポートできます。サポートされている SOP インスタンスと転送構文については、「DICOM サポート」を参照してください。

関連 AWS サービス

AWS HealthImaging は、他の AWS サービスと緊密に統合されています。以下のサービスに関する知識は、 を最大限に活用するのに役立ちます HealthImaging。

- AWS Identity and Access Management – IAM を使用して、ID と HealthImaging リソースへのアクセスを安全に管理します。
- Amazon Simple Storage Service – Amazon S3 をステージングエリアとして使用して、DICOM データを にインポートします HealthImaging。

- Amazon CloudWatch – HealthImaging リソースを監視および監視 CloudWatch するために使用します。
- AWS CloudTrail – HealthImaging ユーザーアクティビティと API の使用状況を追跡 CloudTrail するために使用します。
- AWS CloudFormation – でリソースを作成 AWS CloudFormation するためのコードとしてのインフラストラクチャ (IaC) テンプレートを実装するために使用します HealthImaging。
- AWS PrivateLink — Amazon VPC を使用して、データをインターネットに公開せずに HealthImaging と Amazon Virtual Private Cloud 間の接続を確立します。
- Amazon EventBridge – ターゲット EventBridge にイベントをルーティングするルールを作成して、スケーラブルな HealthImaging イベント駆動型アプリケーションを作成します。

AWS へのアクセス HealthImaging

AWS にアクセスするには HealthImaging AWS Management Console、、 AWS Command Line Interface および AWS SDKsを使用します。このガイドでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。

AWS Management Console

AWS Management Console は、 HealthImaging とそれに関連するリソースを管理するためのウェブベースのユーザーインターフェイスを提供します。 AWS アカウントにサインアップしている場合は、HealthImaging コンソール にサインインできます。

AWS Command Line Interface (AWS CLI)

AWS CLI は、さまざまな AWS 製品用のコマンドを提供し、Windows、Mac、Linux でサポートされています。詳細については、「AWS Command Line Interface ユーザーガイド」を参照してください。

AWS SDKs

AWS SDKsは、ソフトウェア開発者向けのライブラリ、コード例、その他のリソースを提供します。これらのライブラリには、リクエストの暗号化署名、リクエストの再試行、エラーレスポンスの処理などのタスクを自動化する基本的な機能が用意されています。詳細については、「AWS での構築ツール」を参照してください。

HTTP リクエスト

HTTP リクエストを使用して HealthImaging アクションを呼び出すことはできますが、使用するアクションのタイプに応じて異なるエンドポイントを指定する必要があります。詳細については、「HTTP リクエストでサポートされている API アクション」を参照してください。

HIPAA の適格性とデータセキュリティ

これは HIPAA 対象サービスです。 AWS、1996 年米国医療保険の相互運用性と説明責任に関する法律 (HIPAA)、および AWS のサービスを使用して保護医療情報 (PHI) を処理、保存、および送信する方法の詳細については、「HIPAA 概要」を参照してください。

PHI と個人を特定できる情報 (PII) HealthImaging を含む への接続は暗号化する必要があります。デフォルトでは、TLS 経由で HTTPS HealthImaging を使用するすべての接続は、暗号化された顧客コンテンツを HealthImaging 格納し、 AWS 責任共有モデル に従って動作します。

コンプライアンスの詳細については、「AWS HealthImaging のコンプライアンス検証」を参照してください。

料金

HealthImaging は、インテリジェントな階層化により、臨床データのライフサイクル管理を自動化するのに役立ちます。詳細については、「ストレージ階層を理解する」を参照してください。

一般的な料金情報については、「AWS の HealthImaging 料金」を参照してください。コストを見積もるには、AWS HealthImaging 料金計算ツール を使用します。

AWS の開始方法 HealthImaging

AWS の使用を開始するには HealthImaging、 AWS アカウントをセットアップし、 AWS Identity and Access Management ユーザーを作成します。AWS CLI または AWS SDK を使用するには、それらをインストールして設定する必要があります。

HealthImaging 概念とセットアップについて学習した後、使用開始に役立つコード例を含む簡単なチュートリアルを利用できます。

トピック

- AWS HealthImaging の概念
- AWS のセットアップ HealthImaging
- AWS HealthImaging チュートリアル

AWS HealthImaging の概念

AWS HealthImaging を理解して使用する上で重要な用語と概念を以下に示します。

概念

- データストア
- 画像セット
- メタデータ
- 画像フレーム

データストア

データストアは、単一の場所にある医療画像データのリポジトリです。AWS リージョン1つの AWS アカウント内のデータストアはゼロでも複数でもかまいません。各データストアには独自のAWS KMS暗号化キーがあるため、1 つのデータストア内のデータを他のデータストア内のデータから物理的かつ論理的に分離できます。データストアは IAM ロール、権限、属性ベースのアクセス制御によるアクセス制御をサポートします。

詳細については、データストアの管理 および ストレージ階層を理解する を参照してください。

画像セット

画像セットは、関連する医用画像データを最適化するための抽象的なグループ化メカニズムを定義するAWSの概念です。DICOM P10 イメージングデータを AWS HealthImaging データストアにインポートすると、メタデータと画像フレーム(ピクセルデータ) で構成される画像セットに変換されます。DICOM P10 データをインポートすると、同じ DICOM シリーズの 1 つ以上のサービスオブジェクトペア (SOP) インスタンスの DICOM メタデータと画像フレームを含む画像セットが作成されます。

詳細については、画像データのインポート および 画像セットの理解 を参照してください。

メタデータ

メタデータは、画像セット内にある非ピクセル属性です。DICOM の場合、これには患者の人口統計、処置の詳細その他の取得固有パラメーターが含まれます。AWS HealthImaging は、アプリケーションがすばやくアクセスできるように、画像セットをメタデータと画像フレーム (ピクセルデータ) に分離します。これは、ピクセルデータを必要としない画像ビューア、分析、AI/ML のユースケースに役立ちます。DICOM データは患者、治験、シリーズレベルで正規化されるため、不一致がなくなります。これによりデータの使用が容易になり、安全性が向上し、アクセス性能が向上します。

詳細については、画像セットメタデータの取得 および メタデータの正規化 を参照してください。

画像フレーム

画像フレームは、2D 医療画像を構成する画像セット内にあるピクセルデータです。インポート中、AWS HealthImagingはすべての画像フレームを高スループット JPEG 2000 (HTJ2K) でエンコードします。そのため、画像フレームは表示する前にデコードする必要があります。

詳細については、画像セットのピクセルデータの取得 および HTJ2K デコーディングライブラリ を参照してください。

AWS のセットアップ HealthImaging

AWS を使用する前に AWS 環境を設定する必要があります HealthImaging。以下のトピックは、次のセクションにあるチュートリアルの前提条件です。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する

- S3 バケットを作成する
- データストアの作成
- HealthImaging フルアクセス許可を持つ IAM ユーザーを作成する
- インポート用の IAM ロールの作成
- のインストール AWS CLI（オプション）

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

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

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

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

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

管理アクセスを持つユーザーを作成する

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

のセキュリティ保護 AWS アカウントのルートユーザー

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

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

2. ルートユーザーの多要素認証 (MFA) を有効にします。

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

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

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

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

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

管理アクセス権を持つユーザーとしてサインインする

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

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

追加のユーザーにアクセス権を割り当てる

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

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

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

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

S3 バケットを作成する

DICOM P10 データを AWS にインポートするには HealthImaging、2 つの Amazon S3 バケットをお勧めします。Amazon S3 入力バケットは、インポートされる DICOM P10 データを保存し、このバケットから HealthImaging 読み取ります。Amazon S3 出力バケットは、インポートジョブの処理結果を格納し HealthImaging、このバケットに書き込みます。これを視覚的に示した インポートジョブを理解する の図を参照してください。

> **Note**
>
> AWS Identity and Access Management (IAM) ポリシーにより、Amazon S3 バケット名は一意である必要があります。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの名前付け」を参照してください。

このガイドでは、インポート用 IAM ロールに次の Amazon S3 入出力バケットを指定します。

- 入力バケット: `arn:aws:s3:::medical-imaging-dicom-input`
- 出力バケット: `arn:aws:s3:::medical-imaging-output`

詳細については、「Amazon S3 ユーザーガイド」の「バケットの作成」を参照してください。

データストアの作成

医療画像データをインポートすると、AWS HealthImaging データストアは画像セット と呼ばれる変換された DICOM P10 ファイルの結果を保持します。これを視覚的に示した インポートジョブを理解する の図を参照してください。

> **Tip**
>
> `datastoreID` はデータストアを作成すると生成されます。trust relationship が完了すると、このセクションの後半で説明するインポートで `datastoreID` を使用する必要があります。

データストアを作成するにはデータストアの作成を参照してください。

HealthImaging フルアクセス許可を持つ IAM ユーザーを作成する

> **ベストプラクティス**
>
> インポート、データアクセス、データ管理などのさまざまなニーズに合わせて、個別の IAM ユーザーを作成することをお勧めします。これはAWS Well-Architected フレームワークの最小特権アクセスの付与と整合します。
> 次のセクションのチュートリアルでは、1 人の IAM ユーザーを使用します。

IAM ユーザーを作成するには

1. 「IAM ユーザーガイド」の AWS 「アカウントで IAM ユーザーを作成する」の手順に従います。わかりやすくするため、ユーザー `ahiadmin` (または同様のもの) などの命名法を検討してください。
2. `AWSHealthImagingFullAccess` 管理ポリシーを IAM ユーザーに適用します。詳細については、「AWS マネージドポリシー: AWSHealthImagingFullAccess」を参照してください。

   > **Note**
   >
   > IAM の権限は絞り込むことができます。詳細については、「AWS HealthImaging の AWS マネージドポリシー」を参照してください。

インポート用の IAM ロールの作成

> **Note**
>
> 次の手順は、DICOM データをインポートするための Amazon S3 バケットへの読み取りおよび書き込みアクセスを許可する AWS Identity and Access Management (IAM) ロールを参照します。このロールは次のセクションのチュートリアルで必須ですが、AWS HealthImaging の AWS マネージドポリシー を使ってユーザー、グループ、ロールに IAM アクセス権限を追加することをお勧めします。その方が自分でポリシーを作成するより簡単です。

IAM ロールは、特定の許可があり、 アカウントで作成できる IAM アイデンティティです。インポートジョブを開始するには、StartDICOMImportJob アクションを呼び出す IAM ロールを、DICOM P10 データの読み取りとインポートジョブの処理結果の保存に使用する Amazon S3 バケットへのアクセスを許可するユーザーポリシーにアタッチする必要があります。また、AWS がロールを引き受けることができる信頼関係 (ポリシー) HealthImaging を割り当てる必要があります。

インポート用に IAM ロールを作成する

1. IAM コンソールを使用して、ImportJobDataAccessRole の名称を持つ IAM ロールを作成します。このロールは、次のセクションのチュートリアルで使用します。詳細については、「IAM ユーザーガイド」の「IAM ロールの作成」を参照してください。

   > **Tip**
   >
   > このガイドでは、インポートジョブの開始 のコード例は ImportJobDataAccessRole IAM ロールを参考にしています。

2. この IAM ロールに次の IAM アクセス許可ポリシーをアタッチします。このアクセス許可ポリシーは Amazon S3 入出力バケットへのアクセス権を付与します。次の IAM アクセス権限ポリシーをこの IAM ロール ImportJobDataAccessRole にアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::medical-imaging-dicom-input",
                "arn:aws:s3:::medical-imaging-output"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::medical-imaging-dicom-input/*"
            ],
```

```
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::medical-imaging-output/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

3. ImportJobDataAccessRole IAM ロールに以下の信頼関係を追加します。信頼ポリシーでは、データストアの作成 セクションの完了時に生成された datastoreId が必要です。このトピックのチュートリアルでは、1 つの AWS HealthImaging データストアを使用しているが、データストア固有の Amazon S3 バケット、IAM ロール、信頼ポリシーを使用していることを前提としています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "medical-imaging.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:SourceAccount": "accountId"
                },
                "ForAllValues:ArnEquals": {
                    "aws:SourceArn": "arn:aws:medical-
imaging:region:accountId:datastore/datastoreId"
                }
            }
        }
    ]
}
```

AWS での IAM ポリシーの作成と使用の詳細については HealthImaging、「」を参照してくださいAWS の Identity and Access Management HealthImaging。

IAM ロールの詳細については、「IAM ユーザーガイド」の「IAM ロール」を参照してください。IAM ポリシーの詳細については、「IAM ユーザーガイド」の「IAM の許可とポリシー」を参照してください。

のインストール AWS CLI （ オプション ）

AWS Command Line Interfaceを使用している場合は、以下の手順が必要です。 AWS Management Console または AWS SDKsを使用している場合は、次の手順をスキップできます。

を設定するには AWS CLI

1. AWS CLIをダウンロードして設定します。手順については、AWS Command Line Interface ユーザーガイド の次のトピックを参照してください。
   - の最新バージョンのインストールまたは更新 AWS CLI
   - の開始方法 AWS CLI
2. AWS CLI `config` ファイルで、管理者の名前付きプロファイルを追加します。このプロファイルは、 AWS CLI コマンドを実行するときに使用します。最小特権というセキュリティ原則のもと、実行中のタスクに固有の権限を持つ IAM ロールを別途作成することをお勧めします。名前付きプロファイルの詳細については、「AWS Command Line Interface ユーザーガイド」の「設定ファイルと認証情報ファイルの設定」を参照してください。

   ```
   [default]
   aws_access_key_id = default access key ID
   aws_secret_access_key = default secret access key
   region = region
   ```
3. 次の `help` コマンドを使用して設定を確認します。

   ```
   aws medical-imaging help
   ```

   が正しく設定されている場合 AWS CLI は、AWS HealthImaging の簡単な説明と使用可能なコマンドのリストが表示されます。

AWS HealthImaging チュートリアル

目的

このチュートリアルの目的は、DICOM P10 ファイルを AWS HealthImaging データストアにインポートし、メタデータと画像フレーム (ピクセルデータ) で構成される画像セットに変換することです。DICOM データをインポートしたら、 へのアクセス設定に基づいて画像セット、メタデータ、画像フレームにアクセスします HealthImaging。

前提条件

このチュートリアルを完了するには、「設定」 に記載されている手順がすべて必要です。

チュートリアルのステップ

1. インポートジョブを開始する
2. インポートジョブのプロパティを取得する
3. 画像セットを検索する
4. 画像セットのプロパティを取得する
5. 画像セットのメタデータを取得する
6. 画像セットのピクセルデータを取得する
7. データストアを削除する

AWS によるデータストアの管理 HealthImaging

AWS では HealthImaging、医療画像リソースのデータストアを作成および管理します。以下のトピックでは、、、 AWS SDKs を使用して、 HealthImaging アクションを使用してデータストアを作成、説明 AWS Management Console、一覧表示 AWS CLI、削除する方法について説明します。

> **Note**
>
> この章の最後のトピックは、ストレージ階層を理解することです。医療画像データをデータストアにインポートすると、時間と使用状況に基づいて 2 つのストレージ階層間で自動的に移動します。ストレージ階層の料金レベルは異なるため、階層の移動プロセスと請求目的で認識される HealthImaging リソースを理解することが重要です。

トピック

データストアの作成

`CreateDatastore` アクションを使用して、DICOM P10 ファイルをインポートするための AWS HealthImaging データストアを作成します。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスの`CreateDatastore`「」を参照してください。

> **[重要]**
>
> 保護対象の医療情報 (PHI)、個人を特定できる情報 (PII)、またはその他の機密情報や秘匿性の高い情報はデータストア名に使用しないでください。

データストアを作成するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアの作成ページを開きます。
2. [詳細] の [データストア名] に、データストアの名前を入力します。
3. 「データ暗号化」で、リソースを暗号化するための AWS KMS キーを選択します。詳細については、「AWS でのデータ保護 HealthImaging」を参照してください。
4. [タグ - オプション] では、データストアを作成するときに、データストアにタグを追加できます。詳細については、「リソースのタギング」を参照してください。
5. [データストアの作成] を選択します。

AWS CLI および SDKs

Bash

AWS CLI Bash スクリプトを使用する

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function imaging_create_datastore
#
# This function creates an AWS HealthImaging data store for importing DICOM P10
 files.
#
# Parameters:
#       -n data_store_name - The name of the data store.
#
# Returns:
#       The datastore ID.
#    And:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function imaging_create_datastore() {
  local datastore_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_create_datastore"
    echo "Creates an AWS HealthImaging data store for importing DICOM P10 files."
    echo "  -n data_store_name - The name of the data store."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) datastore_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$datastore_name" ]]; then
    errecho "ERROR: You must provide a data store name with the -n parameter."
    usage
    return 1
  fi

  response=$(aws medical-imaging create-datastore \
    --datastore-name "$datastore_name" \
    --output text \
    --query 'datastoreId')

  local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports medical-imaging create-datastore operation
 failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- API の詳細については、「 コマンドリファレンスCreateDatastore」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアを作成するには

次の create-datastore コード例では、my-datastore という名が付けられたデータストアを作成しています。

```
aws medical-imaging create-datastore \
    --datastore-name "my-datastore"
```

出力:

```
{
    "datastoreId": "12345678901234567890123456789012",
    "datastoreStatus": "CREATING"
}
```

詳細については、「 デベロッパーガイド」の「データストアの作成」を参照してください。 AWS HealthImaging

- API の詳細については、「 コマンドリファレンスCreateDatastore」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static String createMedicalImageDatastore(MedicalImagingClient
medicalImagingClient,
           String datastoreName) {
       try {
           CreateDatastoreRequest datastoreRequest =
CreateDatastoreRequest.builder()
                   .datastoreName(datastoreName)
                   .build();
           CreateDatastoreResponse response =
medicalImagingClient.createDatastore(datastoreRequest);
           return response.datastoreId();
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return "";
   }
```

- API の詳細については、「 API リファレンスCreateDatastore」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- API の詳細については、「 API リファレンスCreateDatastore」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def create_datastore(self, name):
        """
        Create a data store.

        :param name: The name of the data store to create.
        :return: The data store ID.
        """
        try:
            data_store =
 self.health_imaging_client.create_datastore(datastoreName=name)
        except ClientError as err:
            logger.error(
                "Couldn't create data store %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return data_store["datastoreId"]
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 CreateDatastore AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

データストアのプロパティの取得

`GetDatastore` アクションを使用して、AWS HealthImaging データストアのプロパティを取得します。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスの`GetDatastore`「」を参照してください。

データストアのプロパティを取得するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   [データストアの詳細] ページが開きます。[詳細] セクションには、すべてのデータストアのプロパティが表示されます。関連する画像セット、インポート、タグを表示するには、該当するタブを選択します。

AWS CLI および SDKs

Bash

AWS CLI Bash スクリプトを使用する

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}
```

```
##############################################################################
# function imaging_get_datastore
#
# Get a data store's properties.
#
# Parameters:
#       -i data_store_id - The ID of the data store.
#
# Returns:
#       [datastore_name, datastore_id, datastore_status, datastore_arn,
 created_at, updated_at]
#    And:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function imaging_get_datastore() {
  local datastore_id option OPTARG # Required to use getopts command in a
 function.
  local error_code
  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_get_datastore"
    echo "Gets a data store's properties."
    echo "  -i datastore_id - The ID of the data store."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "i:h" option; do
    case "${option}" in
      i) datastore_id="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1
```

```
  if [[ -z "$datastore_id" ]]; then
    errecho "ERROR: You must provide a data store ID with the -i parameter."
    usage
    return 1
  fi

  local response

  response=$(
    aws medical-imaging get-datastore \
      --datastore-id "$datastore_id" \
      --output text \
      --query "[ datastoreProperties.datastoreName,
 datastoreProperties.datastoreId, datastoreProperties.datastoreStatus,
 datastoreProperties.datastoreArn,  datastoreProperties.createdAt,
 datastoreProperties.updatedAt]"
  )
  error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-datastores operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- API の詳細については、「 コマンドリファレンスGetDatastore」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアのプロパティを取得するには

次の get-datastore コード例では、データストアのプロパティを取得しています。

```
aws medical-imaging get-datastore \
    --datastore-id 12345678901234567890123456789012
```

出力:

```
{
    "datastoreProperties": {
        "datastoreId": "12345678901234567890123456789012",
        "datastoreName": "TestDatastore123",
        "datastoreStatus": "ACTIVE",
        "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
        "createdAt": "2022-11-15T23:33:09.643000+00:00",
        "updatedAt": "2022-11-15T23:33:09.643000+00:00"
    }
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「データストアプロパティの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetDatastore」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static DatastoreProperties
 getMedicalImageDatastore(MedicalImagingClient medicalImagingClient,
            String datastoreID) {
        try {
            GetDatastoreRequest datastoreRequest = GetDatastoreRequest.builder()
                    .datastoreId(datastoreID)
                    .build();
```

```
            GetDatastoreResponse response =
 medicalImagingClient.getDatastore(datastoreRequest);
            return response.datastoreProperties();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API の詳細については、「 API リファレンスGetDatastore」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
```

```
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxx:datastore/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
};
```

- API の詳細については、「 API リファレンスGetDatastore」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def get_datastore_properties(self, datastore_id):
        """
        Get the properties of a data store.

        :param datastore_id: The ID of the data store.
        :return: The data store properties.
        """
```

```
        try:
            data_store = self.health_imaging_client.get_datastore(
                datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't get data store %s. Here's why: %s: %s",
                id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return data_store["datastoreProperties"]
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
client = boto3.client("medical-imaging")
medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 GetDatastore AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

データストアの一覧表示

ListDatastores アクションを使用して、AWS で利用可能なデータストアを一覧表示します HealthImaging。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスのListDatastores「」を参照してください。

データストアを一覧表示するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

- HealthImaging コンソールのデータストアページを開きます。

  すべてのデータストアは [データストア] セクションに一覧表示されます。

AWS CLI および SDKs

Bash

AWS CLI Bash スクリプトを使用する

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function imaging_list_datastores
#
# List the HealthImaging data stores in the account.
#
# Returns:
#       [[datastore_name, datastore_id, datastore_status]]
#    And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function imaging_list_datastores() {
  local option OPTARG # Required to use getopts command in a function.
  local error_code
  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_list_datastores"
    echo "Lists the AWS HealthImaging data stores in the account."
    echo ""
  }
```

```
  # Retrieve the calling parameters.
  while getopts "h" option; do
    case "${option}" in
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  local response
  response=$(aws medical-imaging list-datastores \
    --output text \
    --query "datastoreSummaries[*][datastoreName, datastoreId, datastoreStatus]")
  error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-datastores operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- API の詳細については、「 コマンドリファレンスListDatastores」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアを一覧表示するには

次の `list-datastores` コード例では、利用可能なデータストアを一覧表示しています。

```
aws medical-imaging list-datastores
```

出力:

```
{
    "datastoreSummaries": [
        {
            "datastoreId": "12345678901234567890123456789012",
            "datastoreName": "TestDatastore123",
            "datastoreStatus": "ACTIVE",
            "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
            "createdAt": "2022-11-15T23:33:09.643000+00:00",
            "updatedAt": "2022-11-15T23:33:09.643000+00:00"
        }
    ]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「データストアの一覧表示」を参照してください。

- API の詳細については、「 コマンドリファレンスListDatastores」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static List<DatastoreSummary>
 listMedicalImagingDatastores(MedicalImagingClient medicalImagingClient) {
        try {
            ListDatastoresRequest datastoreRequest =
 ListDatastoresRequest.builder()
                    .build();
```

```
            ListDatastoresIterable responses =
 medicalImagingClient.listDatastoresPaginator(datastoreRequest);
            List<DatastoreSummary> datastoreSummaries = new ArrayList<>();

            responses.stream().forEach(response ->
 datastoreSummaries.addAll(response.datastoreSummaries()));

            return datastoreSummaries;
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API の詳細については、「 API リファレンスListDatastores」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);
```

```
  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- API の詳細については、「 API リファレンスListDatastores」の「」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_datastores(self):
        """
        List the data stores.

        :return: The list of data stores.
        """
        try:
            paginator =
 self.health_imaging_client.get_paginator("list_datastores")
            page_iterator = paginator.paginate()
            datastore_summaries = []
            for page in page_iterator:
                datastore_summaries.extend(page["datastoreSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't list data stores. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return datastore_summaries
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 ListDatastores AWS SDK for Python (Boto3) API リファレンス の「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

データストアの削除

`DeleteDatastore` アクションを使用して、AWS HealthImaging データストア を削除します。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスの`DeleteDatastore`「」を参照してください。

> **Note**
>
> データストアを削除する前に、まずデータストア内のすべての画像セットを削除する必要があります。詳細については、「画像セットの削除」を参照してください。

データストアを削除する

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。
3. [削除] を選択します。

   [データストアの削除] ページが開きます。
4. データストアの削除を確認するには、テキスト入力フィールドにデータストア名を入力します。
5. [データストアを削除] を選択します。

AWS CLI および SDKs

Bash

AWS CLI Bash スクリプトを使用する

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function imaging_delete_datastore
#
# This function deletes an AWS HealthImaging data store.
#
# Parameters:
#       -i datastore_id - The ID of the data store.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function imaging_delete_datastore() {
  local datastore_id response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_delete_datastore"
    echo "Deletes an AWS HealthImaging data store."
    echo "  -i datastore_id - The ID of the data store."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "i:h" option; do
    case "${option}" in
      i) datastore_id="${OPTARG}" ;;
```

```
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$datastore_id" ]]; then
    errecho "ERROR: You must provide a data store ID with the -i parameter."
    usage
    return 1
  fi

  response=$(aws medical-imaging delete-datastore \
    --datastore-id "$datastore_id")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports medical-imaging delete-datastore operation
 failed.$response"
    return 1
  fi

  return 0
}
```

- API の詳細については、「 コマンドリファレンスDeleteDatastore」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアを削除するには

次の delete-datastore コード例では、データストアを削除しています。

```
aws medical-imaging delete-datastore \
    --datastore-id "12345678901234567890123456789012"
```

出力:

```
{
    "datastoreId": "12345678901234567890123456789012",
    "datastoreStatus": "DELETING"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「データストアの削除」を参照してください。

- API の詳細については、「 コマンドリファレンスDeleteDatastore」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void deleteMedicalImagingDatastore(MedicalImagingClient
medicalImagingClient,
           String datastoreID) {
       try {
           DeleteDatastoreRequest datastoreRequest =
DeleteDatastoreRequest.builder()
                   .datastoreId(datastoreID)
                   .build();
           medicalImagingClient.deleteDatastore(datastoreRequest);
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「 API リファレンスDeleteDatastore」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //       },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスDeleteDatastore」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def delete_datastore(self, datastore_id):
        """
        Delete a data store.

        :param datastore_id: The ID of the data store.
        """
        try:
            self.health_imaging_client.delete_datastore(datastoreId=datastore_id)
        except ClientError as err:
            logger.error(
                "Couldn't delete data store %s. Here's why: %s: %s",
                datastore_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 DeleteDatastore AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ストレージ階層を理解する

AWS HealthImaging は、臨床ライフサイクルの自動管理にインテリジェントな階層化を使用します。これにより、新規またはアクティブなデータと長期アーカイブデータの両方に対して、摩擦ゼロで魅力的なパフォーマンスと価格が得られます。次の階層を使用して、GB/月あたりのストレージに HealthImaging 請求されます。

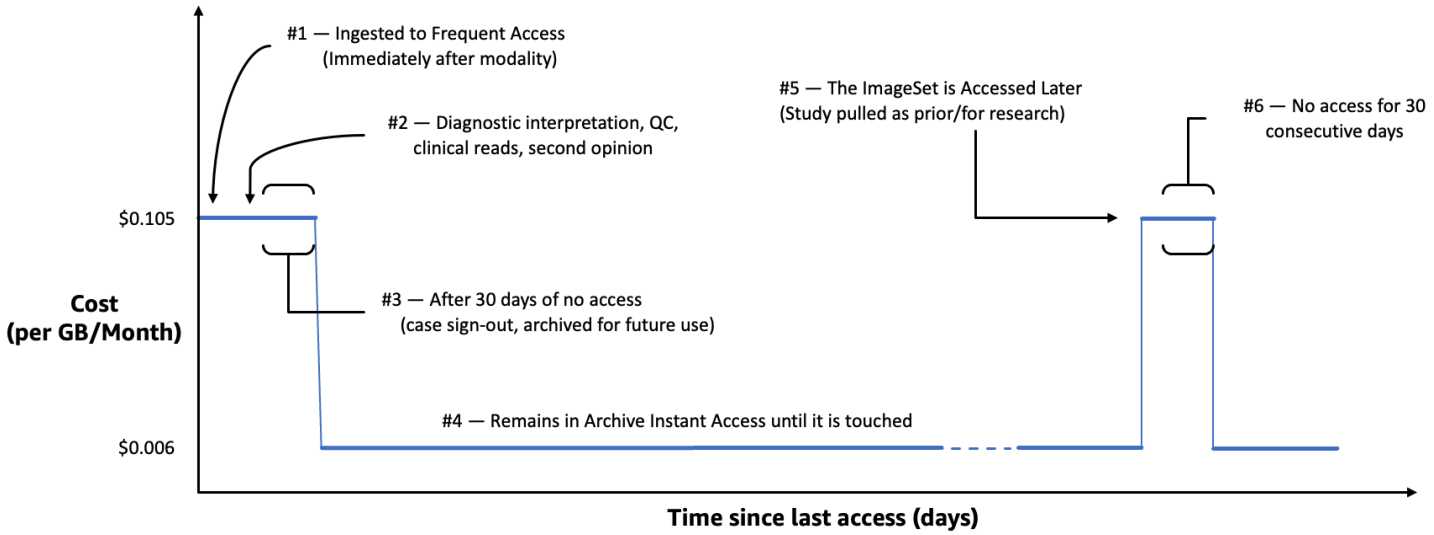
- 高頻度アクセス階層 — アクセス頻度の高いデータ用の階層。
- アーカイブインスタントアクセス階層 — アーカイブされたデータ用の階層。

> **Note**
>
> 高頻度アクセス階層とアーカイブインスタントアクセス階層のパフォーマンスに違いはありません。インテリジェント階層化は特定の画像セットの API アクションに適用されます。インテリジェント階層化は、データストア、インポート、タグ付けの API アクションを認識しません。階層間の移動は API の使用状況に基づいて自動的に行われます。これについては次のセクションで説明します。

階層の移動の仕組み

- インポート後、画像セットは高頻度アクセス階層で開始されます。
- 30 日間連続して何も操作しないと 、画像セットは自動的にアーカイブインスタントアクセス階層に移動します。
- アーカイブインスタントアクセス階層の画像セットは、操作された後にのみ高頻度アクセス階層に戻ります。

次のグラフは、 HealthImaging インテリジェント階層化プロセスの概要を示しています。



操作とみなされるもの

タッチは、 AWS Management Console、、または AWS SDKsであり AWS CLI、次の場合に発生します。

1. 新しい画像セットが作成される (`StartDICOMImportJob` または `CopyImageSet`)
2. 画像セットが更新される (`UpdateImageSetMetadata` または `CopyImageSet`)
3. 画像セットに関連するメタデータまたはイメージフレーム (ピクセルデータ) が読み込まれる (`GetImageSetMetaData` または `GetImageFrame`)

次の HealthImaging API アクションでは、イメージセットをアーカイブインスタントアクセス階層から高頻度アクセス階層にタッチして移動します。

- `StartDICOMImportJob`
- `GetImageSetMetadata`
- `GetImageFrame`
- `CopyImageSet`
- `UpdateImageSetMetadata`

> **Note**
>
> イメージフレーム (ピクセルデータ) は UpdateImageSetMetadata アクションでは削除できませんが、請求目的のためカウントされます。

次の HealthImaging API アクションでは、タッチは行われません。そのため、画像セットはアーカイブインスタントアクセス階層から高頻度アクセス階層に移動しません。

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- GetDICOMImportJob
- ListDICOMImportJobs
- SearchImageSets
- GetImageSet
- ListImageSetVersions
- DeleteImageSet
- TagResource
- ListTagsForResource
- UntagResource

AWS HealthImaging による画像データのインポート

インポートとは、医療画像データを Amazon S3 入力バケットから AWS HealthImaging データストアに移動するプロセスです。インポート中、AWS HealthImaging はピクセルデータの検証チェックを実行してから、DICOM P10 ファイルをメタデータとイメージフレーム (ピクセルデータ) で構成される画像セットに変換します。

> **Tip**
>
> HealthImaging に慣れたら、ぜひ AWS HealthImaging のサンプルプロジェクト にアクセスして、インポートと表示のプロジェクトを使用して実装をすぐに始められるようにすることをお勧めします。

以下のトピックでは、AWS Management Console、AWS CLI、AWS SDK を使用して医療画像データを HealthImaging データストアにインポートする方法について説明します。

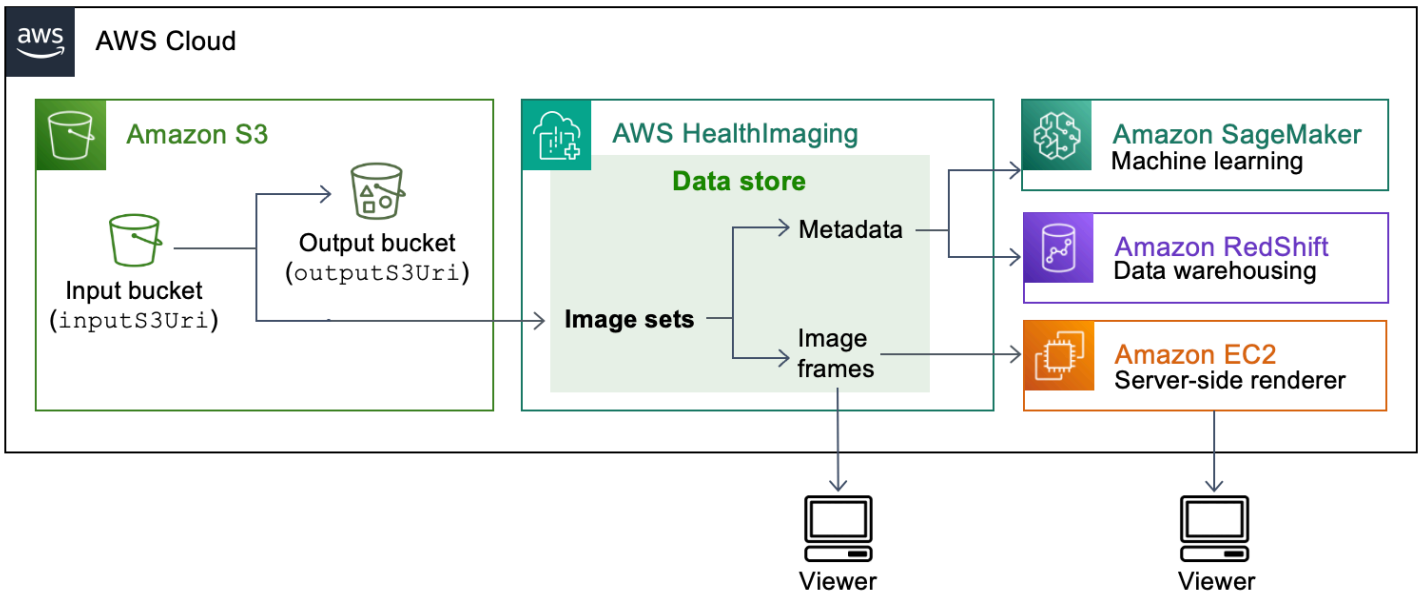
トピック

- インポートジョブを理解する
- インポートジョブの開始
- インポートジョブプロパティの取得
- インポートジョブの一覧表示

インポートジョブを理解する

AWS でデータストアを作成したら HealthImaging、Amazon S3 入力バケットからデータストアに医療画像データをインポートして、画像セット を作成する必要があります。 AWS Management Console、 AWS CLI、 AWS SDKs を使用して、インポートジョブを開始、説明、一覧表示できます。

次の図は、 が DICOM データをデータストアに HealthImaging インポートし、画像セットに変換する方法の概要を示しています。インポートジョブの処理結果は Amazon S3 出力バケット (`outputS3Uri`) に保存され、画像セットは AWS HealthImaging データストアに保存されます。



Amazon S3 から AWS HealthImaging データストアに医療画像ファイルをインポートするときは、次の点に注意してください。

- インポートジョブでは、特定の SOP クラスと転送構文がサポートされています。詳細については、「DICOM サポート」を参照してください。
- インポート中、特定の DICOM 要素には長さの制限が適用されます。インポートジョブを正常に実行するには、医療画像データが長さの制限を超えていないことを確認してください。詳細については、「DICOM 要素の制約」を参照してください。
- インポートジョブの開始時に、ピクセルデータの検証チェックが実行されます。詳細については、「ピクセルデータ検証」を参照してください。
- HealthImaging インポートアクションには、エンドポイント、クォータ、スロットリングの制限があります。詳細については、「エンドポイントとクォータ」および「スロットリングの制限」を参照してください。
- インポートジョブごとに、処理結果は `outputS3Uri` の場所に保存されます。処理結果は、`job-output-manifest.json`ファイル、`SUCCESS` および `FAILURE` フォルダで整理されます。

> **Note**
>
> 1 つのインポートジョブに最大 10,000 個の入れ子になったフォルダーを含めることができます。

- この job-output-manifest.json ファイルには、jobSummary 出力と処理されたデータに関する追加情報が含まれます。次の例は、job-output-manifest.json ファイルからの出力を示しています。

```
{
    "jobSummary": {
        "jobId": "09876543210987654321098765432109",
        "datastoreId": "12345678901234567890123456789012",
        "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",
        "outputS3Uri": "s3://medical-imaging-output/
job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/",
        "successOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/SUCCESS/",
        "failureOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/FAILURE/",
        "numberOfScannedFiles": 5,
        "numberOfImportedFiles": 3,
        "numberOfFilesWithCustomerError": 2,
        "numberOfFilesWithServerError": 0,
        "numberOfGeneratedImageSets": 2,
        "imageSetsSummary": [{
                "imageSetId": "12345612345612345678907890789012",
                "numberOfMatchedSOPInstances": 2
            },
            {
                "imageSetId": "123456123456123456789017891789012",
                "numberOfMatchedSOPInstances": 1
            }
        ]
    }
}
```

- この SUCCESS フォルダには、正常にインポートされたすべての画像ファイルの結果を含む success.ndjson ファイルが格納されます。次の例は、success.ndjson ファイルからの出力を示しています。

```
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105620.1.0.1.dcm","importResponse
{"imageSetId":"12345612345612345678907890789012"}}
```

```
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105630.1.0.1.dcm","importResponse
{"imageSetId":"12345612345612345678917891789012"}}
```

- この FAILURE フォルダには、正常にインポートされなかったすべての画像ファイルの結果を含む failure.ndjson ファイルが格納されます。次の例は、failure.ndjson ファイルからの出力を示しています。

```
{"inputFile":"dicom_input/invalidDicomFile1.dcm","exception":
{"exceptionType":"ValidationException","message":"DICOM attribute TransferSyntaxUID
 does not exist"}}
{"inputFile":"dicom_input/invalidDicomFile2.dcm","exception":
{"exceptionType":"ValidationException","message":"DICOM attributes does not
 exist"}}
```

- インポートジョブは 90 日間ジョブのリストに保持され、その後アーカイブされます。

インポートジョブの開始

StartDICOMImportJob アクションを使用して、ピクセルデータ検証チェックと AWS データ HealthImaging ストアへの一括データインポートを開始します。インポートジョブは、inputS3Uri パラメータで指定された Amazon S3 入力バケットにある DICOM P10 ファイルをインポートします。インポートジョブの処理結果は、outputS3Uri パラメータで指定された Amazon S3 出力バケットに保存されます。

> **Note**
>
> HealthImaging は、サポートされている他のリージョン にある Amazon S3 バケットからのデータインポートをサポートします。この機能を実現するには、インポートジョブを開始するときに inputOwnerAccountIdパラメータを指定します。詳細については、「のクロスアカウントインポート AWS HealthImaging」を参照してください。
> インポート中、特定の DICOM 要素には長さの制限が適用されます。詳細については、「DICOM 要素の制約」を参照してください。

次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスのStartDICOMImportJob「」を参照してください。

インポートジョブを開始するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。
3. [DICOM データをインポート] を選択します。

   [DICOM データをインポート] ページが開きます。
4. 詳細 セクションで、次の情報を入力します。
   - 名前 (オプション)
   - S3 でソースの場所をインポートする
   - ソースバケット所有者のアカウント ID (オプション)
   - 暗号化キー (オプション)
   - S3 の出力先
5. [サービスアクセス] セクションで [既存のサービスロールを使用する] を選択し、[サービスロール名] メニューからロールを選択するか、[新しいサービスロールを作成して使用する] を選択します。
6. Import (インポート) を選択します。

AWS CLI および SDKs

C++

SDK for C++

```
//! Routine which starts a HealthImaging import job.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param inputBucketName: The name of the Amazon S3 bucket containing the DICOM
 files.
```

```
  \param inputDirectory: The directory in the S3 bucket containing the DICOM
 files.
  \param outputBucketName: The name of the S3 bucket for the output.
  \param outputDirectory: The directory in the S3 bucket to store the output.
  \param roleArn: The ARN of the IAM role with permissions for the import.
  \param importJobId: A string to receive the import job ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::startDICOMImportJob(
        const Aws::String &dataStoreID, const Aws::String &inputBucketName,
        const Aws::String &inputDirectory, const Aws::String &outputBucketName,
        const Aws::String &outputDirectory, const Aws::String &roleArn,
        Aws::String &importJobId,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(clientConfig);
    Aws::String inputURI = "s3://" + inputBucketName + "/" + inputDirectory +
 "/";
    Aws::String outputURI = "s3://" + outputBucketName + "/" + outputDirectory +
 "/";
    Aws::MedicalImaging::Model::StartDICOMImportJobRequest
 startDICOMImportJobRequest;
    startDICOMImportJobRequest.SetDatastoreId(dataStoreID);
    startDICOMImportJobRequest.SetDataAccessRoleArn(roleArn);
    startDICOMImportJobRequest.SetInputS3Uri(inputURI);
    startDICOMImportJobRequest.SetOutputS3Uri(outputURI);

    Aws::MedicalImaging::Model::StartDICOMImportJobOutcome
 startDICOMImportJobOutcome = medicalImagingClient.StartDICOMImportJob(
            startDICOMImportJobRequest);

    if (startDICOMImportJobOutcome.IsSuccess()) {
        importJobId = startDICOMImportJobOutcome.GetResult().GetJobId();
    }
    else {
        std::cerr << "Failed to start DICOM import job because "
                  << startDICOMImportJobOutcome.GetError().GetMessage() <<
 std::endl;
    }

    return startDICOMImportJobOutcome.IsSuccess();
}
```

- API の詳細については、「 API リファレンス」のStartDICOMImportJob」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

DICOM インポートジョブを開始するには

次の `start-dicom-import-job` コード例では、DICOM インポートジョブを開始しています。

```
aws medical-imaging start-dicom-import-job \
    --job-name "my-job" \
    --datastore-id "12345678901234567890123456789012" \
    --input-s3-uri "s3://medical-imaging-dicom-input/dicom_input/" \
    --output-s3-uri "s3://medical-imaging-output/job_output/" \
    --data-access-role-arn "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole"
```

出力:

```
{
    "datastoreId": "12345678901234567890123456789012",
    "jobId": "09876543210987654321098765432109",
    "jobStatus": "SUBMITTED",
    "submittedAt": "2022-08-12T11:28:11.152000+00:00"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「インポートジョブの開始」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StartDICOMImportJob」を参照してください。

Java

SDK for Java 2.x

```
   public static String startDicomImportJob(MedicalImagingClient
medicalImagingClient,
           String jobName,
           String datastoreId,
           String dataAccessRoleArn,
           String inputS3Uri,
           String outputS3Uri) {

       try {
           StartDicomImportJobRequest startDicomImportJobRequest =
StartDicomImportJobRequest.builder()
                   .jobName(jobName)
                   .datastoreId(datastoreId)
                   .dataAccessRoleArn(dataAccessRoleArn)
                   .inputS3Uri(inputS3Uri)
                   .outputS3Uri(outputS3Uri)
                   .build();
           StartDicomImportJobResponse response =
medicalImagingClient.startDICOMImportJob(startDicomImportJobRequest);
           return response.jobId();
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return "";
   }
```

- API の詳細については、「 API リファレンス」のStartDICOMImportJob」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input
 files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files
 are stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
```

```
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- API の詳細については、「 API リファレンス」のStartDICOMImportJob」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def start_dicom_import_job(
        self, job_name, datastore_id, role_arn, input_s3_uri, output_s3_uri
    ):
        """
        Start a DICOM import job.

        :param job_name: The name of the job.
        :param datastore_id: The ID of the data store.
        :param role_arn: The Amazon Resource Name (ARN) of the role to use for
 the job.
        :param input_s3_uri: The S3 bucket input prefix path containing the DICOM
 files.
        :param output_s3_uri: The S3 bucket output prefix path for the result.
        :return: The job ID.
        """
        try:
```

```
            job = self.health_imaging_client.start_dicom_import_job(
                jobName=job_name,
                datastoreId=datastore_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_s3_uri,
                outputS3Uri=output_s3_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
client = boto3.client("medical-imaging")
medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 AWS SDK for StartDICOMImportJob」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

インポートジョブプロパティの取得

GetDICOMImportJob アクションを使用して、AWS HealthImaging インポートジョブのプロパティの詳細を確認します。たとえば、インポートジョブを開始した後に、GetDICOMImportJob を実行してジョブのステータスを確認できます。jobStatus が COMPLETED に戻ったら、画像セットにアクセスする準備は完了です。

> **Note**
>
> jobStatus はインポートジョブの実行を指します。そのため、インポート処理中に検証の問題が見つかった場合でも、インポートジョブは jobStatus を COMPLETED と返すことがあります。jobStatus が COMPLETED として返される場合でも、Amazon S3 に書き込まれた出力マニフェストを確認することをお勧めします。マニフェストには、個々の P10 オブジェクトのインポートの成功または失敗に関する詳細が記載されるためです。

次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスのGetDICOMImportJob「」を参照してください。

インポートジョブのプロパティを取得するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   [データストアの詳細] ページが開きます。[画像セット] タブはデフォルトで選択されています。
3. [インポート] タブを選択します。
4. インポートジョブを選択します。

   [インポートジョブの詳細] ページが開き、インポートジョブに関するプロパティが表示されます。

AWS CLI および SDKs

C++

SDK for C++

```
//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
  \param clientConfig: Aws client configuration.
```

```
  \return GetDICOMImportJobOutcome: The import job outcome.
*/
Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
AwsDoc::Medical_Imaging::getDICOMImportJob(const Aws::String &dataStoreID,
                                           const Aws::String &importJobID,
                                           const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetDICOMImportJobRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetJobId(importJobID);
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome outcome =
 client.GetDICOMImportJob(
            request);
    if (!outcome.IsSuccess()) {
        std::cerr << "GetDICOMImportJob error: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome;
}
```

- API の詳細については、「 API リファレンスGetDICOMImportJob」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

DICOM インポートジョブのプロパティを取得するには

次の `get-dicom-import-job` コード例では、DICOM インポートジョブのプロパティを取得しています。

```
aws medical-imaging get-dicom-import-job \
```

```
    --datastore-id "12345678901234567890123456789012" \
    --job-id "09876543210987654321098765432109"
```

出力:

```
{
    "jobProperties": {
        "jobId": "09876543210987654321098765432109",
        "jobName": "my-job",
        "jobStatus": "COMPLETED",
        "datastoreId": "12345678901234567890123456789012",
        "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
        "endedAt": "2022-08-12T11:29:42.285000+00:00",
        "submittedAt": "2022-08-12T11:28:11.152000+00:00",
        "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",
        "outputS3Uri": "s3://medical-imaging-output/
job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/"
    }
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「インポートジョブのプロパティの取得」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GetDICOMImportJob」を参照してください。

Java

SDK for Java 2.x

```
   public static DICOMImportJobProperties getDicomImportJob(MedicalImagingClient
 medicalImagingClient,
            String datastoreId,
            String jobId) {

        try {
            GetDicomImportJobRequest getDicomImportJobRequest =
 GetDicomImportJobRequest.builder()
                    .datastoreId(datastoreId)
                    .jobId(jobId)
```

```
                    .build();
            GetDicomImportJobResponse response =
 medicalImagingClient.getDICOMImportJob(getDicomImportJobRequest);
            return response.jobProperties();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API の詳細については、「 API リファレンスGetDICOMImportJob」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
```

```
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスGetDICOMImportJob」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client
```

```
    def get_dicom_import_job(self, datastore_id, job_id):
        """
        Get the properties of a DICOM import job.

        :param datastore_id: The ID of the data store.
        :param job_id: The ID of the job.
        :return: The job properties.
        """
        try:
            job = self.health_imaging_client.get_dicom_import_job(
                jobId=job_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't get DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobProperties"]
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 AWS SDK for GetDICOMImportJob」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

インポートジョブの一覧表示

`ListDICOMImportJobs` アクションを使用して、特定の HealthImaging データストア用に作成されたインポートジョブを一覧表示します。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスの`ListDICOMImportJobs`「」を参照してください。

> **Note**
>
> インポートジョブは 90 日間ジョブのリストに保持され、その後アーカイブされます。

インポートジョブを一覧表示するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   [データストアの詳細] ページが開きます。[画像セット] タブはデフォルトで選択されています。
3. [インポート] タブを選択すると、関連するすべてのインポートジョブが表示されます。

AWS CLI および SDKs

CLI

AWS CLI

DICOM インポートジョブを一覧表示するには

次の `list-dicom-import-jobs` コード例では、インポートジョブを一覧表示します。

```
aws medical-imaging list-dicom-import-jobs \
    --datastore-id "12345678901234567890123456789012"
```

出力:

```
{
```

```
    "jobSummaries": [
        {
            "jobId": "09876543210987654321098765432109",
            "jobName": "my-job",
            "jobStatus": "COMPLETED",
            "datastoreId": "12345678901234567890123456789012",
            "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
            "endedAt": "2022-08-12T11:21:56.504000+00:00",
            "submittedAt": "2022-08-12T11:20:21.734000+00:00"
        }
    ]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「インポートジョブの一覧表示」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ListDICOMImportJobs」を参照してください。

Java

SDK for Java 2.x

```
   public static List<DICOMImportJobSummary>
listDicomImportJobs(MedicalImagingClient medicalImagingClient,
            String datastoreId) {

        try {
            ListDicomImportJobsRequest listDicomImportJobsRequest =
ListDicomImportJobsRequest.builder()
                    .datastoreId(datastoreId)
                    .build();
            ListDicomImportJobsResponse response =
medicalImagingClient.listDICOMImportJobs(listDicomImportJobsRequest);
            return response.jobSummaries();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return new ArrayList<>();
    }
```

- API の詳細については、「 API リファレンスListDICOMImportJobs」を参照してください。AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
```

```
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/
dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- API の詳細については、「 API リファレンスListDICOMImportJobs」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_dicom_import_jobs(self, datastore_id):
```

```
        """
        List the DICOM import jobs.

        :param datastore_id: The ID of the data store.
        :return: The list of jobs.
        """
        try:
            paginator = self.health_imaging_client.get_paginator(
                "list_dicom_import_jobs"
            )
            page_iterator = paginator.paginate(datastoreId=datastore_id)
            job_summaries = []
            for page in page_iterator:
                job_summaries.extend(page["jobSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't list DICOM import jobs. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job_summaries
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 AWS SDK for Python (Boto3) API リファレンスの「ListDICOMImportJobs」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS による画像セットへのアクセス HealthImaging

AWS の医療画像データにアクセスするには、 HealthImaging 通常、一意のキーを持つ画像セットを検索し、関連するメタデータと画像フレーム (ピクセルデータ) を取得する必要があります。

> **Tip**
>
> AWS に慣れたら HealthImaging、AWS HealthImaging のサンプルプロジェクト「」にアクセスして、表示プロジェクトを使用して実装をすぐに開始することをお勧めします。

以下のトピックでは、画像セットとは何か AWS Management Console、、 AWS CLI、 AWS SDKs を使用して画像セットを検索し、関連するプロパティ、メタデータ、画像フレームを取得する方法について説明します。

トピック

- 画像セットの理解
- 画像セットの検索
- 画像セットのプロパティの取得
- 画像セットメタデータの取得
- 画像セットのピクセルデータの取得
- DICOM インスタンスの取得

画像セットの理解

画像セットは、AWS の基盤となる AWS 概念です HealthImaging。画像セットは DICOM データをにインポートするときに作成されるため HealthImaging、サービスを使用する際には画像セットを十分に理解する必要があります。

画像セットが導入された理由は次のとおりです。

- 柔軟な API により、さまざまな医療画像ワークフロー (臨床および非臨床) をサポートします。
- 関連データのみをグループ化することで、患者の安全性を最大限に高めることができます。
- 不一致の可視性を高めるために、データのクリーニングを促します。詳細については、「画像セットの変更」を参照してください。

> [重要]
> クリーニング前に DICOM データを臨床使用すると、患者に害を及ぼす可能性があります。

以下のメニューでは、画像セットについて詳しく説明し、 でその機能と目的を理解するのに役立つ例と図を示します HealthImaging。

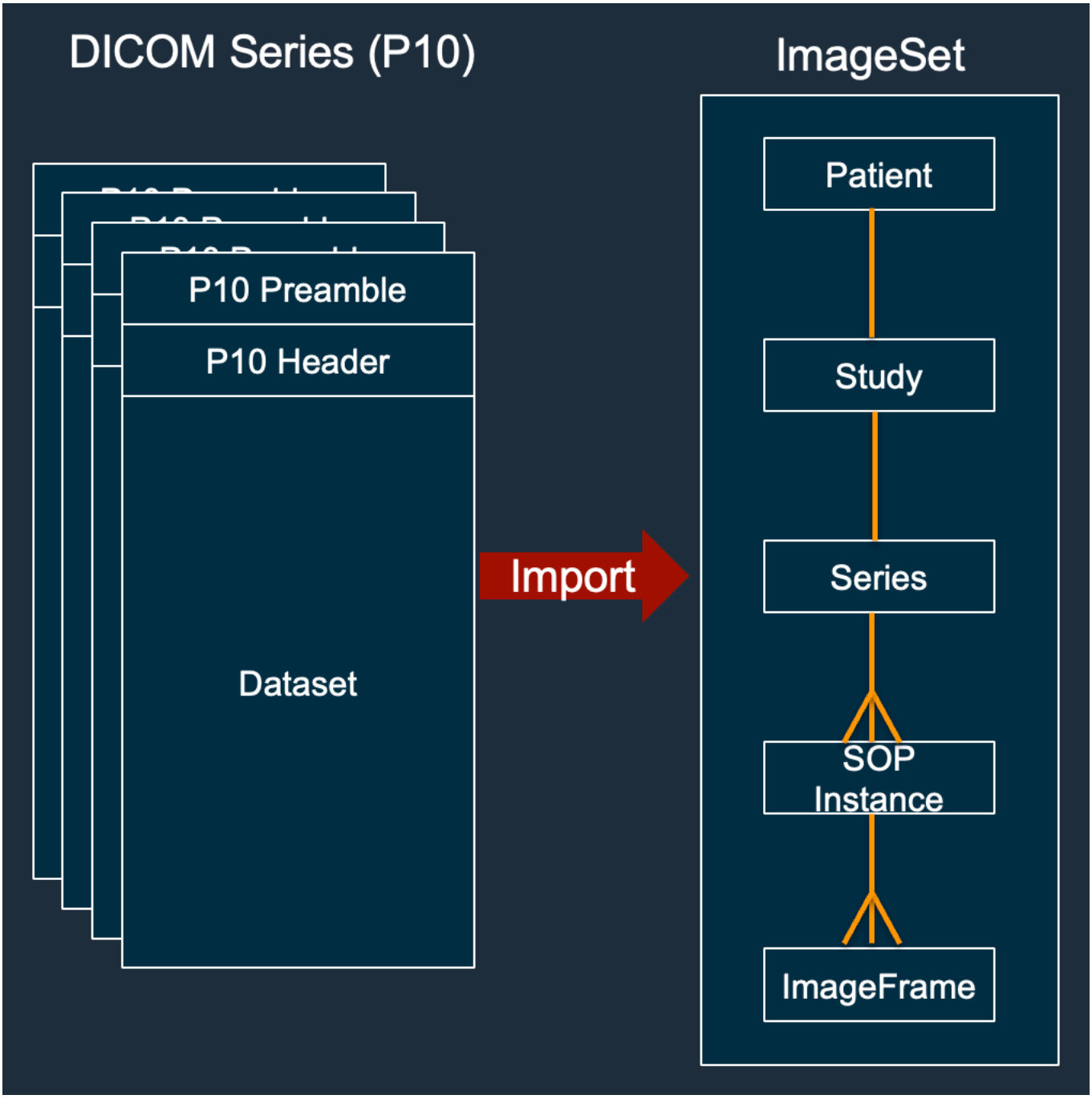
画像セットとは

画像セットは、関連する医療画像データを最適化するための抽象的なグループ化メカニズムを定義する AWS 概念です。DICOM P10 画像データを AWS HealthImaging データストアにインポートすると、メタデータと画像フレーム (ピクセルデータ) で構成される画像セットに変換されます。

> Note
> 画像セットのメタデータは正規化されています。つまり、共通の属性と値のセットの 1 つが、DICOM データ要素レジストリにリストされている患者、研究、シリーズレベルの要素に対応しているということです。
> 画像フレーム (ピクセルデータ) はハイスループット JPEG 2000 (HTJ2K) でエンコードされるため、表示される前にデコードする必要があります。

画像セットは AWS リソースであるため、Amazon リソースネーム (ARNs)が割り当てられます。キーと値のペアを 50 個までタグ付けでき、IAM を通じてロールベースのアクセス制御 (RBAC) と属性ベースのアクセス制御 (ABAC) が可能です。さらに、画像セットはバージョン管理されているため、すべての変更は保存され、以前のバージョンにもアクセスできます。

DICOM P10 データをインポートすると、同じ DICOM シリーズの 1 つ以上のサービスオブジェクトペア (SOP) インスタンスの DICOM メタデータと画像フレームを含む画像セットが作成されます。



Note

DICOM インポートジョブ:

- 常に新しい画像セットを作成し、既存の画像セットは更新しないでください。
- 同じ SOP インスタンスをインポートするたびに追加のストレージが使用されるため、SOP インスタンスのストレージを重複排除しないでください。

- 1 つの DICOM シリーズに対して複数の画像セットを作成する場合があります。例えば、PatientName不一致などの正規化されたメタデータ属性のバリアントがある場合です。

画像セットメタデータはどのようなものですか？

GetImageSetMetadata アクションを使用して、画像セットメタデータを取得します。返されるメタデータは で圧縮されるためgzip、表示する前に解凍する必要があります。詳細については、「画像セットメタデータの取得」を参照してください。

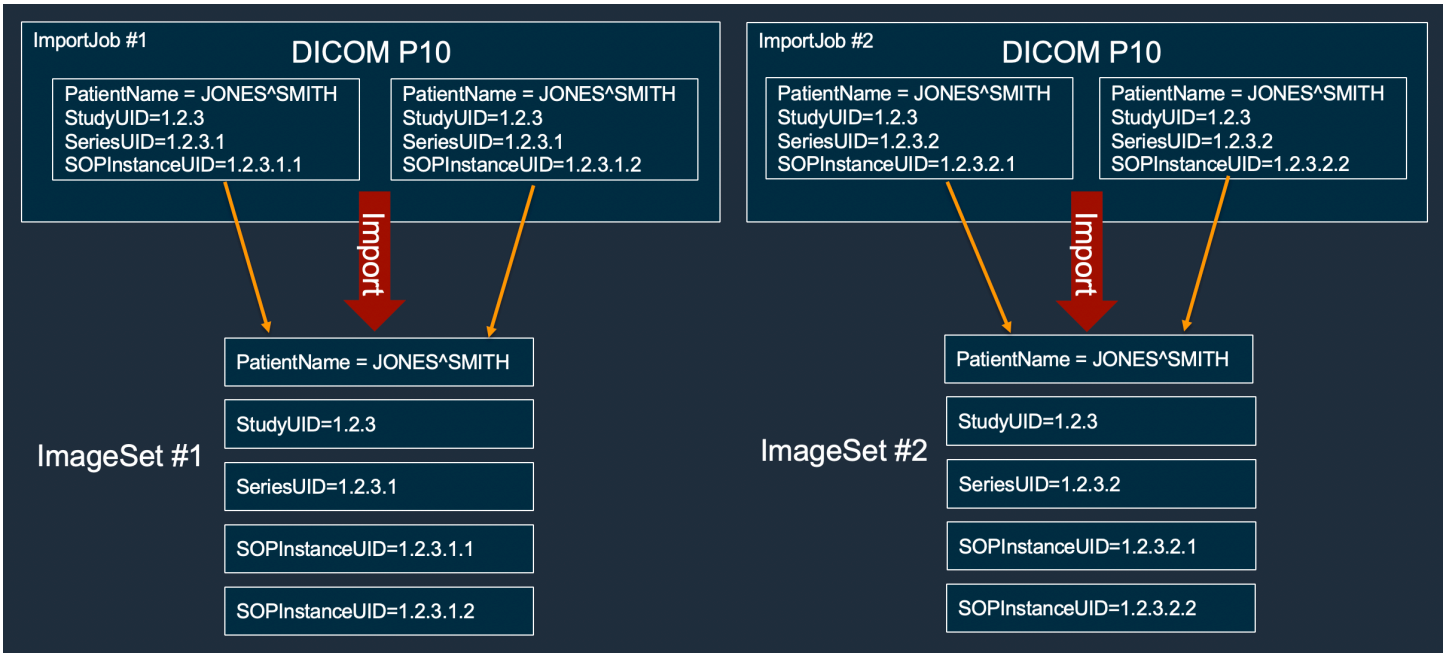
次の例は、JSON 形式の画像セットメタデータの構造を示しています。

```
{
 "SchemaVersion": "1.1",
 "DatastoreID": "2aa75d103f7f45ab977b0e93f00e6fe9",
 "ImageSetID": "46923b66d5522e4241615ecd64637584",
 "Patient": {
  "DICOM": {
   "PatientBirthDate": null,
   "PatientSex": null,
   "PatientID": "2178309",
   "PatientName": "MISTER^CT"
  }
 },
 "Study": {
  "DICOM": {
   "StudyTime": "083501",
   "PatientWeight": null
  },
  "Series": {
   "1.2.840.113619.2.30.1.1762295590.1623.978668949.887": {
    "DICOM": {
        "Modality": "CT",
     "PatientPosition": "FFS"
    },
    "Instances": {
     "1.2.840.113619.2.30.1.1762295590.1623.978668949.888": {
      "DICOM": {
       "SourceApplicationEntityTitle": null,
       "SOPClassUID": "1.2.840.10008.5.1.4.1.1.2",
       "HighBit": 15,
```

```
        "PixelData": null,
        "Exposure": "40",
        "RescaleSlope": "1",
       "ImageFrames": [
        {
         "ID": "0d1c97c51b773198a3df44383a5fd306",
         "PixelDataChecksumFromBaseToFullResolution": [
          {
           "Width": 256,
           "Height": 188,
           "Checksum": 2598394845
          },
          {
           "Width": 512,
           "Height": 375,
           "Checksum": 1227709180
          }
         ],
         "MinPixelValue": 451,
         "MaxPixelValue": 1466,
         "FrameSizeInBytes": 384000
        }
       ]
      }
     }
    }
   }
  }
 }
}
```

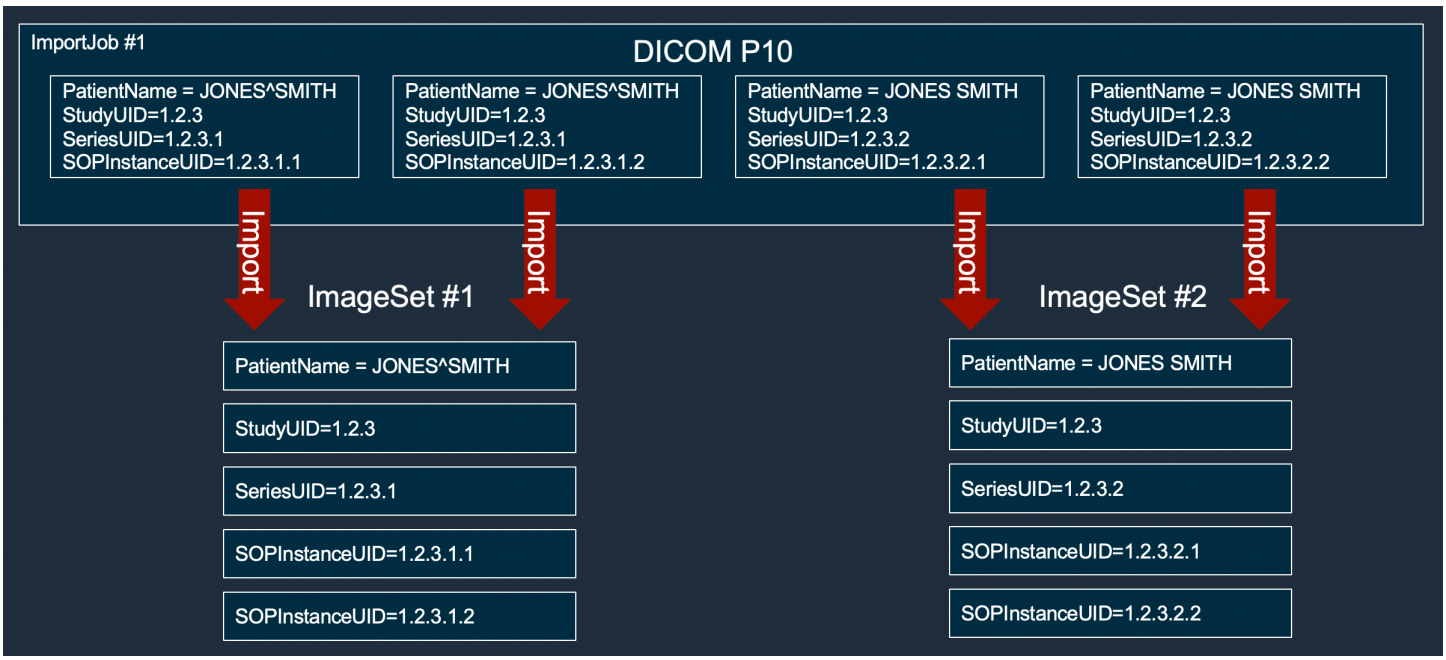
画像セットの作成例: 複数のインポートジョブ

次の例は、複数のインポートジョブが常に新しいイメージセットを作成し、既存のイメージセットに絶対に追加しないことを示しています。



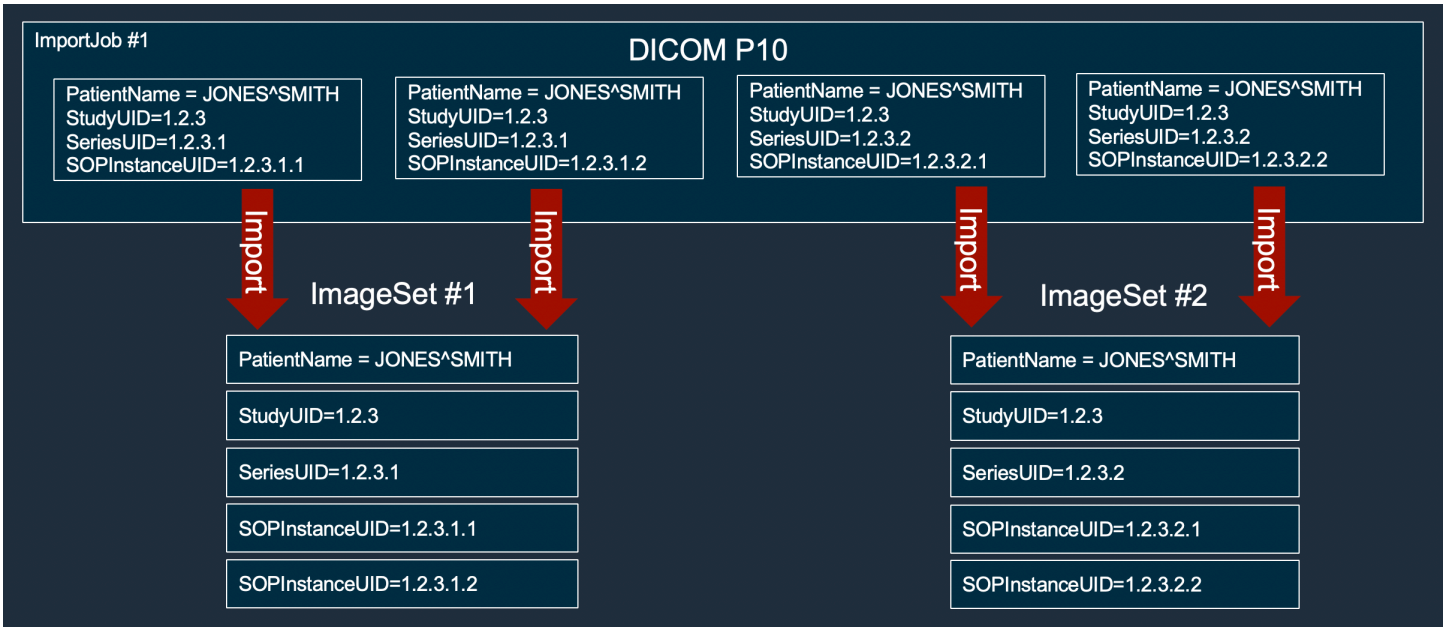
画像セットの作成例: 2 つのバリアントを含む 1 つのインポートジョブ

次の例は、インスタンス 1 と 2 の患者名がインスタンス 3 と 4 と異なるために、1 つのインポートジョブで 2 つの画像セットが作成されることを示しています。



画像セットの作成例: 最適化を含む単一のインポートジョブ

以下の例は、患者名が一致していても 1 つのインポートジョブで 2 つの画像セットを作成して、スループットを向上させる様子を示しています。



画像セットの検索

`SearchImageSets` アクションを使用して、`ACTIVE` HealthImaging データストア内のすべての画像セットに対して検索クエリを実行します。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスの`SearchImageSets`「」を参照してください。

> **Note**
>
> 画像セットを検索するときは、次の点に注意してください。
>
> - `SearchImageSets` は 1 つの検索クエリパラメータを受け入れ、条件に一致するすべての画像セットについて、ページ分割されたレスポンスを返します。すべての日付範囲クエリは として入力する必要があります`(lowerBound, upperBound)`。
> - デフォルトでは、 `SearchImageSets`は `updatedAt`フィールドを使用して、最新から古い順でソートします。
> - 顧客所有の AWS KMS キーを使用してデータストアを作成した場合は、イメージセットを操作する前に AWS KMS キーポリシーを更新する必要があります。詳細については、「カスタマーマネージドキーの作成」を参照してください。

画像セットを検索するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

> **Note**
>
> 次の手順は、 `Series Instance UID`および `Updated at`プロパティフィルターを使用して画像セットを検索する方法を示しています。

Series Instance UID

`Series Instance UID` プロパティフィルターを使用して画像セットを検索する

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   データストアの詳細ページが開き、デフォルトで [画像セット] タブが選択されます。
3. プロパティフィルターメニューを選択し、 を選択しますSeries Instance UID。
4. 検索する値の入力フィールドに、目的のシリーズインスタンス UID を入力します (貼り付け) 。

   > **Note**
   >
   > シリーズインスタンスの UID 値は、DICOM 一意識別子のレジストリ (UIDs) に記載されている値と同じである必要があります。要件には、少なくとも 1 つの期間を含む一連の数値が含まれていることに注意してください。期間 は、シリーズインスタンス UIDs の先頭または末尾では使用できません。文字と空白は許可されないため、UIDs をコピーして貼り付けるときは注意が必要です。
5. 日付範囲メニューを選択し、シリーズインスタンス UID の日付範囲を選択し、適用 を選択します。
6. [検索] を選択します。

   選択した日付範囲内にあるシリーズインスタンス UIDs は、デフォルトで最新の順序で返されます。

Updated at

Updated at プロパティフィルターを使用して画像セットを検索する

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   データストアの詳細ページが開き、デフォルトで [画像セット] タブが選択されます。
3. プロパティフィルターメニューを選択し、 を選択しますUpdated at。
4. 日付範囲メニューを選択し、画像セットの日付範囲を選択し、適用 を選択します。
5. [検索] を選択します。

   選択した日付範囲内の画像セットは、デフォルトで最新の順序で返されます。

AWS CLI および SDKs

C++

SDK for C++

イメージセットを検索するためのユーティリティ関数。

```
//! Routine which searches for image sets based on defined input attributes.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param searchCriteria: A search criteria instance.
  \param imageSetResults: Vector to receive the image set IDs.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::searchImageSets(const Aws::String &dataStoreID,
                                              const
 Aws::MedicalImaging::Model::SearchCriteria &searchCriteria,
                                              Aws::Vector<Aws::String>
 &imageSetResults,
                                              const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::SearchImageSetsRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetSearchCriteria(searchCriteria);
```

```
    Aws::String nextToken; // Used for paginated results.
    bool result = true;
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::MedicalImaging::Model::SearchImageSetsOutcome outcome =
 client.SearchImageSets(
                request);
        if (outcome.IsSuccess()) {
            for (auto &imageSetMetadataSummary:
 outcome.GetResult().GetImageSetsMetadataSummaries()) {

 imageSetResults.push_back(imageSetMetadataSummary.GetImageSetId());
            }

            nextToken = outcome.GetResult().GetNextToken();
        }
        else {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            result = false;
        }
    } while (!nextToken.empty());

    return result;
}
```

ユースケース #1: EQUAL 演算子。

```
        Aws::Vector<Aws::String> imageIDsForPatientID;
        Aws::MedicalImaging::Model::SearchCriteria searchCriteriaEqualsPatientID;
        Aws::Vector<Aws::MedicalImaging::Model::SearchFilter>
 patientIDSearchFilters = {

 Aws::MedicalImaging::Model::SearchFilter().WithOperator(Aws::MedicalImaging::Model::Oper

 .WithValues({Aws::MedicalImaging::Model::SearchByAttributeValue().WithDICOMPatientId(pat
        };
```

```
        searchCriteriaEqualsPatientID.SetFilters(patientIDSearchFilters);
        bool result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,

 searchCriteriaEqualsPatientID,

 imageIDsForPatientID,
                                                                clientConfig);
        if (result) {
            std::cout << imageIDsForPatientID.size() << " image sets found for
 the patient with ID '"
            <<  patientID << "'." << std::endl;
            for (auto &imageSetResult : imageIDsForPatientID) {
                std::cout << "  Image set with ID '" << imageSetResult <<
 std::endl;
            }
        }
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
         Aws::MedicalImaging::Model::SearchByAttributeValue useCase2StartDate;

 useCase2StartDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAnd
        .WithDICOMStudyDate("19990101")
        .WithDICOMStudyTime("000000.000"));

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase2EndDate;

 useCase2EndDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAndTi

 .WithDICOMStudyDate(Aws::Utils::DateTime(std::chrono::system_clock::now()).ToLocalTimeSt
%m%d"))
        .WithDICOMStudyTime("000000.000"));

        Aws::MedicalImaging::Model::SearchFilter useCase2SearchFilter;
        useCase2SearchFilter.SetValues({useCase2StartDate, useCase2EndDate});

 useCase2SearchFilter.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchCriteria useCase2SearchCriteria;
        useCase2SearchCriteria.SetFilters({useCase2SearchFilter});
```

```
        Aws::Vector<Aws::String> usesCase2Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase2SearchCriteria,
                                                          usesCase2Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase2Results.size() << " image sets found for
between 1999/01/01 and present."
                      <<  std::endl;
            for (auto &imageSetResult : usesCase2Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
        Aws::MedicalImaging::Model::SearchByAttributeValue useCase3StartDate;

useCase3StartDate.SetCreatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase3EndDate;

useCase3EndDate.SetCreatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));

        Aws::MedicalImaging::Model::SearchFilter useCase3SearchFilter;
        useCase3SearchFilter.SetValues({useCase3StartDate, useCase3EndDate});

useCase3SearchFilter.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchCriteria useCase3SearchCriteria;
        useCase3SearchCriteria.SetFilters({useCase3SearchFilter});

        Aws::Vector<Aws::String> usesCase3Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase3SearchCriteria,
                                                          usesCase3Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase3Results.size() << " image sets found for
created between 2023/11/30 and present."
```

```
                    <<  std::endl;
            for (auto &imageSetResult : usesCase3Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順でのソートレスポンス。

```
        Aws::MedicalImaging::Model::SearchByAttributeValue useCase4StartDate;

useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase4EndDate;

useCase4EndDate.SetUpdatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterBetween;
        useCase4SearchFilterBetween.SetValues({useCase4StartDate,
useCase4EndDate});

useCase4SearchFilterBetween.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchByAttributeValue seriesInstanceUID;
        seriesInstanceUID.SetDICOMSeriesInstanceUID(dicomSeriesInstanceUID);

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterEqual;
        useCase4SearchFilterEqual.SetValues({seriesInstanceUID});

useCase4SearchFilterEqual.SetOperator(Aws::MedicalImaging::Model::Operator::EQUAL);

        Aws::MedicalImaging::Model::SearchCriteria useCase4SearchCriteria;
        useCase4SearchCriteria.SetFilters({useCase4SearchFilterBetween,
useCase4SearchFilterEqual});

        Aws::MedicalImaging::Model::Sort useCase4Sort;

useCase4Sort.SetSortField(Aws::MedicalImaging::Model::SortField::updatedAt);
        useCase4Sort.SetSortOrder(Aws::MedicalImaging::Model::SortOrder::ASC);

        useCase4SearchCriteria.SetSort(useCase4Sort);
```

```
        Aws::Vector<Aws::String> usesCase4Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase4SearchCriteria,
                                                          usesCase4Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase4Results.size() << " image sets found for EQUAL
operator "
            << "on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort
response\n"
            <<  "in ASC order on updatedAt field." <<  std::endl;
            for (auto &imageSetResult : usesCase4Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

- API の詳細については、「 API リファレンスSearchImageSets」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

例 1 : EQUAL 演算子を使用して画像セットを検索するには

次の `search-image-sets` コード例では、EQUAL 演算子を使用し、特定の値に基づいて画像セットを検索しています。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria file://search-criteria.json
```

`search-criteria.json` の内容

```
{
    "filters": [{
        "values": [{"DICOMPatientId" : "SUBJECT08701"}],
        "operator": "EQUAL"
    }]
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
             "DICOMPatientSex": "F",
             "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
             "DICOMPatientBirthDate": "19201120",
             "DICOMStudyDescription": "UNKNOWN",
             "DICOMPatientId": "SUBJECT08701",
             "DICOMPatientName": "Melissa844 Huel628",
             "DICOMNumberOfStudyRelatedInstances": 1,
             "DICOMStudyTime": "140728",
             "DICOMNumberOfStudyRelatedSeries": 1
            },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

例 2: DICOM StudyDate と DICOM を使用して BETWEEN 演算子で画像セットを検索するにはStudyTime

次の `search-image-sets` コード例では、1990 年 1 月 1 日 (午前 0 時) から 2023 年 1 月 1 日 (午前 0 時) の間に生成された DICOM スタディを含む画像セットを検索します。

注: DICOM StudyTime はオプションです。入力されていない場合は、フィルターで指定された日付の時間値は午前 0 時 (1 日の始まり) になります。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
```

```
    --search-criteria file://search-criteria.json
```

search-criteria.json の内容

```
{
    "filters": [{
        "values": [{
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "19900101",
                "DICOMStudyTime": "000000"
            }
        },
        {
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "20230101",
                "DICOMStudyTime": "000000"
            }
        }],
        "operator": "BETWEEN"
    }]
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
```

```
    }]
}
```

例 3：createdAt を使用して BETWEEN 演算子を使用して画像セットを検索するには (スタディが以前に保存されていた時間)

次のsearch-image-setsコード例では、UTC タイムゾーンの時間範囲内に DICOM スタディ HealthImaging が保持されている画像セットを検索します。

注：createdAt をサンプル形式 ("1985-04-12T23:20:50.52Z") で提供してください。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria  file://search-criteria.json
```

search-criteria.json の内容

```
{
    "filters": [{
        "values": [{
            "createdAt": "1985-04-12T23:20:50.52Z"
        },
        {
            "createdAt": "2022-04-12T23:20:50.52Z"
        }],
        "operator": "BETWEEN"
    }]
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
```

```
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

例 4: DICOM SeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN を使用して画像セットを検索し、updatedAt フィールドで ASC 順序でレスポンスをソートするには

次のsearch-image-setsコード例では、DICOM SeriesInstanceUID に EQUAL 演算子を使用し、updatedAt に BETWEEN を使用し、updatedAt フィールドに ASC 順でレスポンスをソートする画像セットを検索します。

注: updatedAt をサンプル形式 ("1985-04-12T23:20:50.52Z") で指定します。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria  file://search-criteria.json
```

search-criteria.json の内容

```
{
    "filters": [{
        "values": [{
            "updatedAt": "2024-03-11T15:00:05.074000-07:00"
        }, {
            "updatedAt": "2024-03-11T16:00:05.074000-07:00"
        }],
        "operator": "BETWEEN"
    }, {
        "values": [{
            "DICOMSeriesInstanceUID": "1.2.840.99999999.84710745.943275268089"
        }],
        "operator": "EQUAL"
    }],
    "sort": {
        "sortField": "updatedAt",
        "sortOrder": "ASC"
```

```
    }
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットの検索」を参照してください。

- API の詳細については、「 コマンドリファレンスSearchImageSets」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

イメージセットを検索するためのユーティリティ関数。

```
    public static List<ImageSetsMetadataSummary> searchMedicalImagingImageSets(
            MedicalImagingClient medicalImagingClient,
            String datastoreId, SearchCriteria searchCriteria) {
```

```
        try {
            SearchImageSetsRequest datastoreRequest =
SearchImageSetsRequest.builder()
                    .datastoreId(datastoreId)
                    .searchCriteria(searchCriteria)
                    .build();
            SearchImageSetsIterable responses = medicalImagingClient
                    .searchImageSetsPaginator(datastoreRequest);
            List<ImageSetsMetadataSummary> imageSetsMetadataSummaries = new
ArrayList<>();

            responses.stream().forEach(response -> imageSetsMetadataSummaries
                    .addAll(response.imageSetsMetadataSummaries()));

            return imageSetsMetadataSummaries;
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

ユースケース #1: EQUAL 演算子。

```
        List<SearchFilter> searchFilters =
Collections.singletonList(SearchFilter.builder()
                .operator(Operator.EQUAL)
                .values(SearchByAttributeValue.builder()
                        .dicomPatientId(patientId)
                        .build())
                .build());

        SearchCriteria searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .build();

        List<ImageSetsMetadataSummary> imageSetsMetadataSummaries =
searchMedicalImagingImageSets(
                medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
```

```
            System.out.println("The image sets for patient " + patientId + " are:
\n"
                    + imageSetsMetadataSummaries);
            System.out.println();
        }
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMdd");
        searchFilters = Collections.singletonList(SearchFilter.builder()
                .operator(Operator.BETWEEN)
                .values(SearchByAttributeValue.builder()

 .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                                        .dicomStudyDate("19990101")
                                        .dicomStudyTime("000000.000")
                                        .build())
                                .build(),
                        SearchByAttributeValue.builder()

 .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                                        .dicomStudyDate((LocalDate.now()
                                                .format(formatter)))
                                        .dicomStudyTime("000000.000")
                                        .build())
                                .build())
                .build());

        searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .build();

        imageSetsMetadataSummaries =
 searchMedicalImagingImageSets(medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println(
                    "The image sets searched with BETWEEN operator using
 DICOMStudyDate and DICOMStudyTime are:\n"
                            +
                            imageSetsMetadataSummaries);
            System.out.println();
```

```
        }
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
        searchFilters = Collections.singletonList(SearchFilter.builder()
                .operator(Operator.BETWEEN)
                .values(SearchByAttributeValue.builder()

.createdAt(Instant.parse("1985-04-12T23:20:50.52Z"))
                                .build(),
                        SearchByAttributeValue.builder()
                                .createdAt(Instant.now())
                                .build())
                .build());

        searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .build();
        imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println("The image sets searched with BETWEEN operator
using createdAt are:\n "
                    + imageSetsMetadataSummaries);
            System.out.println();
        }
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順でのソートレスポンス。

```
        Instant startDate = Instant.parse("1985-04-12T23:20:50.52Z");
        Instant endDate = Instant.now();

        searchFilters = Arrays.asList(
                SearchFilter.builder()
                        .operator(Operator.EQUAL)
                        .values(SearchByAttributeValue.builder()
                                .dicomSeriesInstanceUID(seriesInstanceUID)
```

```
                                .build())
                        .build(),
                SearchFilter.builder()
                        .operator(Operator.BETWEEN)
                        .values(

SearchByAttributeValue.builder().updatedAt(startDate).build(),

SearchByAttributeValue.builder().updatedAt(endDate).build()
                        ).build());

        Sort sort =
Sort.builder().sortOrder(SortOrder.ASC).sortField(SortField.UPDATED_AT).build();

        searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .sort(sort)
                .build();

        imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println("The image sets searched with EQUAL operator on
DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response\n" +
                    "in ASC order on updatedAt field are:\n "
                    + imageSetsMetadataSummaries);
            System.out.println();
        }
```

- API の詳細については、「 API リファレンスSearchImageSets」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

イメージセットを検索するためのユーティリティ関数。

```
import {paginateSearchImageSets} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters -
 The search criteria filters.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
 criteria sort.
 */
export const searchImageSets = async (
    datastoreId = "xxxxxxxx",
    searchCriteria = {}
) => {
    const paginatorConfig = {
        client: medicalImagingClient,
        pageSize: 50,
    };

    const commandParams = {
        datastoreId: datastoreId,
        searchCriteria: searchCriteria,
    };

    const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

    const imageSetsMetadataSummaries = [];
    for await (const page of paginator) {
        // Each page contains a list of `jobSummaries`. The list is truncated if
 is larger than `pageSize`.
        imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
        console.log(page);
    }
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
    //         extendedRequestId: undefined,
```

```
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //     imageSetsMetadataSummaries: [
    //         {
    //             DICOMTags: [Object],
    //             createdAt: "2023-09-19T16:59:40.551Z",
    //             imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
    //             updatedAt: "2023-09-19T16:59:40.551Z",
    //             version: 1
    //         }]
    // }

    return imageSetsMetadataSummaries;
};
```

ユースケース #1: EQUAL 演算子。

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [{DICOMPatientId: "1234567"}],
                    operator: "EQUAL",
                },
            ]
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
    const datastoreId = "12345678901234567890123456789012";

    try {
```

```
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {
                            DICOMStudyDateAndTime: {
                                DICOMStudyDate: "19900101",
                                DICOMStudyTime: "000000",
                            },
                        },
                        {
                            DICOMStudyDateAndTime: {
                                DICOMStudyDate: "20230901",
                                DICOMStudyTime: "000000",
                            },
                        },
                    ],
                    operator: "BETWEEN",
                },
            ]
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {createdAt: new Date("1985-04-12T23:20:50.52Z")},
                        {createdAt: new Date()},
                    ],
                    operator: "BETWEEN",
                },
```

```
            ]
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順でのソートレスポンス。

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
                        {updatedAt: new Date()},
                    ],
                    operator: "BETWEEN",
                },
                {
                    values: [
                        {DICOMSeriesInstanceUID:
"1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
                    ],
                    operator: "EQUAL",
                },
            ],
            sort: {
                sortOrder: "ASC",
                sortField: "updatedAt",
            }
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

- API の詳細については、「 API リファレンスSearchImageSets」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

イメージセットを検索するためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
 [{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
 self.health_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries
```

ユースケース #1: EQUAL 演算子。

```
        search_filter = {
            "filters": [
                {"operator": "EQUAL", "values": [{"DICOMPatientId": patient_id}]}
            ]
        }

        image_sets = self.search_image_sets(data_store_id, search_filter)
        print(f"Image sets found with EQUAL operator\n{image_sets}")
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
        search_filter = {
            "filters": [
                {
                    "operator": "BETWEEN",
                    "values": [
                        {
                            "DICOMStudyDateAndTime": {
                                "DICOMStudyDate": "19900101",
                                "DICOMStudyTime": "000000",
                            }
                        },
                        {
                            "DICOMStudyDateAndTime": {
                                "DICOMStudyDate": "20230101",
                                "DICOMStudyTime": "000000",
                            }
                        },
                    ],
                }
            ]
```

```
        }

        image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            f"Image sets found with BETWEEN operator using DICOMStudyDate and
 DICOMStudyTime\n{image_sets}"
        )
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
        search_filter = {
            "filters": [
                {
                    "values": [
                        {
                            "createdAt": datetime.datetime(
                                2021, 8, 4, 14, 49, 54, 429000
                            )
                        },
                        {
                            "createdAt": datetime.datetime.now()
                            + datetime.timedelta(days=1)
                        },
                    ],
                    "operator": "BETWEEN",
                }
            ]
        }

        recent_image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            f"Image sets found with with BETWEEN operator using createdAt
\n{recent_image_sets}"
        )
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順でのソートレスポンス。

```
        search_filter = {
```

```
            "filters": [
                {
                    "values": [
                        {
                            "updatedAt": datetime.datetime(
                                2021, 8, 4, 14, 49, 54, 429000
                            )
                        },
                        {
                            "updatedAt": datetime.datetime.now()
                            + datetime.timedelta(days=1)
                        },
                    ],
                    "operator": "BETWEEN",
                },
                {
                    "values": [{"DICOMSeriesInstanceUID": series_instance_uid}],
                    "operator": "EQUAL",
                },
            ],
            "sort": {
                "sortOrder": "ASC",
                "sortField": "updatedAt",
            },
        }

        image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            "Image sets found with EQUAL operator on DICOMSeriesInstanceUID and
BETWEEN on updatedAt and"
        )
        print(f"sort response in ASC order on updatedAt field\n{image_sets}")
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 SearchImageSets AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

画像セットのプロパティの取得

`GetImageSet` アクションを使用して、 で指定されたイメージセットのプロパティを返します HealthImaging。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスの`GetImageSet`「」を参照してください。

> **Note**
>
> デフォルトでは、AWS はイメージセットの最新バージョンのプロパティ HealthImaging を返します。古いバージョンの画像セットのプロパティを表示するには、 `versionId` をリクエストに入力します。

画像セットのプロパティを取得するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   データストアの詳細ページが開き、デフォルトで [画像セット] タブが選択されます。
3. 画像セットを選択します。

   画像セットの詳細ページが開き、画像セットのプロパティが表示されます。

AWS CLI および SDKs

CLI

AWS CLI

画像セットのプロパティを取得するには

以下の `get-image-set` コード例では、画像セットのプロパティを取得しています。

```
aws medical-imaging get-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id 18f88ac7870584f58d56256646b4d92b \
    --version-id 1
```

出力:

```
{
    "versionId": "1",
    "imageSetWorkflowStatus": "COPIED",
    "updatedAt": 1680027253.471,
    "imageSetId": "18f88ac7870584f58d56256646b4d92b",
    "imageSetState": "ACTIVE",
    "createdAt": 1679592510.753,
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットのプロパティの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetImageSet」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static GetImageSetResponse getMedicalImageSet(MedicalImagingClient
 medicalImagingClient,
            String datastoreId,
            String imagesetId,
            String versionId) {
```

```
        try {
            GetImageSetRequest.Builder getImageSetRequestBuilder =
GetImageSetRequest.builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId);

            if (versionId != null) {
                getImageSetRequestBuilder =
getImageSetRequestBuilder.versionId(versionId);
            }

            return
medicalImagingClient.getImageSet(getImageSetRequestBuilder.build());
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API の詳細については、「 API リファレンスGetImageSet」の「」を参照してください。AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
```

```
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスGetImageSet」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def get_image_set(self, datastore_id, image_set_id, version_id=None):
        """
        Get the properties of an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.health_imaging_client.get_image_set(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set = self.health_imaging_client.get_image_set(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
        except ClientError as err:
            logger.error(
                "Couldn't get image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
        else:
            return image_set
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、GetImageSetAWS 「 SDK for Python (Boto3) API リファレンス」の「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

画像セットメタデータの取得

GetImageSetMetadata アクションを使用して、 で特定のイメージセットのメタデータを取得します HealthImaging。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスのGetImageSetMetadata「」を参照してください。

> **Note**
>
> デフォルトでは、 はイメージセットの最新バージョンのメタデータ属性 HealthImaging を返します。古いバージョンの画像セットのメタデータを表示するには、リクエストにversionIdを付けてください。
> 画像セットのメタデータはgzipで圧縮され、JSON オブジェクトとして返されます。したがって、正規化されたメタデータを表示する前に JSON オブジェクトを解凍する必要があります。詳細については、「メタデータの正規化」を参照してください。

画像セットのメタデータを取得するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   データストアの詳細ページが開き、デフォルトで [画像セット] タブが選択されます。
3. 画像セットを選択します。

   画像セットの詳細ページが開き、画像セットのメタデータが「画像セットメタデータビューア」セクションの下に表示されます。

AWS CLI および SDKs

C++

SDK for C++

イメージセットのメタデータを取得するためのユーティリティ関数。

```
//! Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The HealthImaging image set ID.
  \param versionID: The HealthImaging image set version ID, ignored if empty.
  \param outputFilePath: The path where the metadata will be stored as gzipped
 json.
  \param clientConfig: Aws client configuration.
  \\return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
 &outputFilePath,
                                                  const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
```

```
    if (!versionID.empty()) {
        request.SetVersionId(versionID);
    }
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
 client.GetImageSetMetadata(
            request);
    if (outcome.IsSuccess()) {
        std::ofstream file(outputFilePath, std::ios::binary);
        auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
        file << metadata.rdbuf();
    }
    else {
        std::cerr << "Failed to get image set metadata: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

イメージセットのメタデータをバージョンなしで取得します。

```
        if (AwsDoc::Medical_Imaging::getImageSetMetadata(dataStoreID, imageSetID,
 "", outputFilePath, clientConfig))
        {
            std::cout << "Successfully retrieved image set metadata." <<
 std::endl;
            std::cout << "Metadata stored in: " << outputFilePath << std::endl;
        }
```

イメージセットのメタデータをバージョン付きで取得します。

```
        if (AwsDoc::Medical_Imaging::getImageSetMetadata(dataStoreID, imageSetID,
 versionID, outputFilePath, clientConfig))
        {
            std::cout << "Successfully retrieved image set metadata." <<
 std::endl;
            std::cout << "Metadata stored in: " << outputFilePath << std::endl;
        }
```

- API の詳細については、「 API リファレンスGetImageSetMetadata」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

例 1：画像セットのメタデータをバージョンなしで取得するには

次の get-image-set-metadata コード例では、バージョンを指定せずに画像セットのメタデータを取得しています。

注：outfile は必須のパラメータです

```
aws medical-imaging get-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    studymetadata.json.gz
```

返されたメタデータは gzip で圧縮され、studymetadata.json.gz ファイルに保存されます。返された JSON オブジェクトの内容を表示するには、まずオブジェクトを解凍する必要があります。

出力:

```
{
    "contentType": "application/json",
    "contentEncoding": "gzip"
}
```

例 2：画像セットのメタデータをバージョン付きで取得するには

次の get-image-set-metadata コード例では、指定されたバージョンの画像セットのメタデータを取得しています。

注：outfile は必須のパラメータです

```
aws medical-imaging get-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --version-id 1 \
    studymetadata.json.gz
```

返されたメタデータは gzip で圧縮され、studymetadata.json.gz ファイルに保存されます。返された JSON オブジェクトの内容を表示するには、まずオブジェクトを解凍する必要があります。

出力:

```
{
    "contentType": "application/json",
    "contentEncoding": "gzip"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットメタデータの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetImageSetMetadata」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void getMedicalImageSetMetadata(MedicalImagingClient
medicalImagingClient,
            String destinationPath,
            String datastoreId,
            String imagesetId,
            String versionId) {

        try {
            GetImageSetMetadataRequest.Builder getImageSetMetadataRequestBuilder
= GetImageSetMetadataRequest.builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId);

            if (versionId != null) {
```

```
                getImageSetMetadataRequestBuilder =
getImageSetMetadataRequestBuilder.versionId(versionId);
            }


medicalImagingClient.getImageSetMetadata(getImageSetMetadataRequestBuilder.build(),
                    FileSystems.getDefault().getPath(destinationPath));

            System.out.println("Metadata downloaded to " + destinationPath);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「 API リファレンスGetImageSetMetadata」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

イメージセットのメタデータを取得するためのユーティリティ関数。

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped
 metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
```

```
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

イメージセットのメタデータをバージョンなしで取得します。

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
```

```
      "12345678901234567890123456789012",
      "12345678901234567890123456789012"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

イメージセットのメタデータをバージョン付きで取得します。

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- API の詳細については、「 API リファレンスGetImageSetMetadata」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

イメージセットのメタデータを取得するためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client
```

```
    def get_image_set_metadata(
        self, metadata_file, datastore_id, image_set_id, version_id=None
    ):
        """
        Get the metadata of an image set.

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """
        try:
            if version_id:
                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:

                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
            print(image_set_metadata)
            with open(metadata_file, "wb") as f:
                for chunk in
image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)

        except ClientError as err:
            logger.error(
                "Couldn't get image metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

イメージセットのメタデータをバージョンなしで取得します。

```
                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
```

イメージセットのメタデータをバージョン付きで取得します。

```
                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 GetImageSetMetadata AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

画像セットのピクセルデータの取得

画像フレームは、2D 医療画像を構成する画像セット内にあるピクセルデータです。GetImageFrame アクションを使用して、 で特定の画像セットの HTJ2K-encodedされた画像フレームを取得します HealthImaging。 ???次のメニューは、 AWS CLI および AWS SDKsコード例を示しています。詳細については、AWS HealthImaging API リファレンスのGetImageFrame「」を参照してください。

> **Note**
>
> インポート中、AWS はすべてのイメージフレームを HTJ2K 可逆形式で HealthImaging エンコードするため、イメージビューワーで表示する前にデコードする必要があります。詳細については、「HTJ2K デコーディングライブラリ」を参照してください。

画像フレームを取得するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

> **Note**
>
> AWS Management Consoleには画像ビューアが組み込まれていないため、画像フレームをプログラムでデコードしてアクセスする必要があります。
> 画像フレームのデコードと表示の詳細については、HTJ2K デコーディングライブラリを参照してください。

AWS CLI および SDKs

C++

SDK for C++

```
//! Routine which downloads an AWS HealthImaging image frame.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The image set ID.
  \param frameID: The image frame ID.
  \param jphFile: File to store the downloaded frame.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageFrame(const Aws::String &dataStoreID,
                                            const Aws::String &imageSetID,
                                            const Aws::String &frameID,
                                            const Aws::String &jphFile,
```

```
                                                const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);

    Aws::MedicalImaging::Model::GetImageFrameRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);

    Aws::MedicalImaging::Model::ImageFrameInformation imageFrameInformation;
    imageFrameInformation.SetImageFrameId(frameID);
    request.SetImageFrameInformation(imageFrameInformation);

    Aws::MedicalImaging::Model::GetImageFrameOutcome outcome =
 client.GetImageFrame(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully retrieved image frame." << std::endl;
        auto &buffer = outcome.GetResult().GetImageFrameBlob();

        std::ofstream outfile(jphFile, std::ios::binary);
        outfile << buffer.rdbuf();
    }
    else {
        std::cout << "Error retrieving image frame." <<
 outcome.GetError().GetMessage()
                  << std::endl;

    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「 API リファレンスGetImageFrame」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

画像セットのピクセルデータを取得するには

次の get-image-frame コード例では、画像フレームを取得しています。

```
aws medical-imaging get-image-frame \
    --datastore-id "12345678901234567890123456789012" \
    --image-set-id "98765412345612345678907890789012" \
    --image-frame-information imageFrameId=3abf5d5d7ae72f80a0ec81b2c0de3ef4 \
    imageframe.jph
```

注: GetImageFrame アクションはピクセルデータのストリームを imageframe.jph ファイルに返すため、このコード例には出力は含まれません。画像フレームのデコードと表示については、「HTJ2K デコードライブラリ」を参照してください。

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットのピクセルデータの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetImageFrame」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
       public static void getMedicalImageSetFrame(MedicalImagingClient
medicalImagingClient,
                        String destinationPath,
                        String datastoreId,
                        String imagesetId,
                        String imageFrameId) {

                try {
                        GetImageFrameRequest getImageSetMetadataRequest =
GetImageFrameRequest.builder()
                                        .datastoreId(datastoreId)
                                        .imageSetId(imagesetId)

.imageFrameInformation(ImageFrameInformation.builder()
```

```
                .imageFrameId(imageFrameId)
                                                                .build())
                                                .build();

                medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

                FileSystems.getDefault().getPath(destinationPath));

                                        System.out.println("Image frame downloaded to " +
                destinationPath);
                                } catch (MedicalImagingException e) {
                                        System.err.println(e.awsErrorDetails().errorMessage());
                                        System.exit(1);
                                }
                        }
```

- API の詳細については、「 API リファレンスGetImageFrame」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
```

```
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     contentType: 'application/octet-stream',
  //     imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};
```

- API の詳細については、「 API リファレンスGetImageFrame」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def get_pixel_data(
        self, file_path_to_write, datastore_id, image_set_id, image_frame_id
    ):
        """
        Get an image frame's pixel data.

        :param file_path_to_write: The path to write the image frame's HTJ2K
 encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.health_imaging_client.get_image_frame(
                datastoreId=datastore_id,
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)
        except ClientError as err:
            logger.error(
                "Couldn't get image frame. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 GetImageFrame AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

DICOM インスタンスの取得

> **Note**
>
> API HealthImaging `GetDICOMInstance`は、ウェブベースの医療画像用の DICOMweb Retrieve (WADO-RS) 標準に準拠して構築されています。`GetDICOMInstance` は DICOMweb サービスを表すため、 AWS CLI および AWS SDKsでは提供されません。

`GetDICOMInstance` アクションを使用して、リソースに関連付けられたシリーズ、治験、インスタンス UIDs を指定して、 HealthImaging データストアから DICOM インスタンス (`.dcm` ファイル) を取得します。イメージセット ID をクエリパラメータとして指定することで、インスタンスリソースを取得するイメージセットを指定できます。さらに、非圧縮 (ELE) または高スループット JPEG 2000 (HTJ2K) をサポートする転送構文を選択して DICOM データを圧縮できます。を使用すると`GetDICOMInstance`、 の HealthImagingクラウドネイティブインターフェイスを活用しながら、DICOM Part 10 バイナリを利用するシステムと相互運用できます。

DICOM インスタンスを取得するには (**`.dcm`** ファイル)

1. および `imageSetId`パラメータ値を収集 HealthImaging `datastoreId`します。
2. `GetImageSetMetadata` アクションを `datastoreId`および `imageSetId`パラメータ値とともに使用して`studyInstanceUID`、、`seriesInstanceUID`、および の関連するメタデータ値を取得します`sopInstanceUID`。詳細については、「画像セットメタデータの取得」を参照してください。

3. 、、datastoreId、、および の値を使用してsopInstanceUID、リクエストの URL studyInstanceUID seriesInstanceUIDを作成しますimageSetId。URL は 形式です。

```
https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/
studies/study-instance-uid/series/series-instance-uid/instances/sop-instance-uid?
imageSetId=image-set-id
```

4. リクエストを準備して送信します。 はAWS 署名バージョン 4 の署名プロトコルで HTTP GET リクエストGetDICOMInstanceを使用します。次のコード例では、 curl コマンドラインツールを使用して から DICOM インスタンス (.dcm ファイル) を取得します HealthImaging。

   Shell

```
curl --request GET \
  'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
d9a2a515ab294163a2d2f4069eed584c/
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/
instances/1.3.6.1.4.1.5962.1.1.4.1.1.20040826186059.5457?
imageSetId=459e50687f121185f747b67bb60d1bc8' \
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom; transfer-syntax=1.2.840.10008.1.2.1' \
  --output 'dicom-instance.dcm'
```

> **Note**
>
> transfer-syntax UID はオプションで、含まれていない場合はデフォルトで明示的な VR リトルエンディアンになります。サポートされている転送構文は次のとおりです。
>
> - 明示的な VR リトルエンディアン (ELE) - 1.2.840.10008.1.2.1 (デフォルト)
> - RPCL オプションによる高スループット JPEG 2000 イメージ圧縮 (ロスレスのみ) - 1.2.840.10008.1.2.4.202
>
>   詳細については、「AWS 用の HTJ2K デコードライブラリ HealthImaging」を参照してください。

AWS HealthImaging による画像セットの変更

DICOM インポートジョブでは、通常、以下の理由で画像セットを変更する必要があります。

- 患者の安全
- データ整合性
- ストレージコストの削減

HealthImaging には、画像セットの変更プロセスを簡素化するためにいくつかの API が用意されています。以下のトピックでは、AWS CLI および AWS SDK を使用して画像セットを変更する方法について説明します。

トピック

画像セットのバージョンを一覧表示する

`ListImageSetVersions` アクションを使用して、 で画像セットのバージョン履歴を一覧表示します HealthImaging。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスの`ListImageSetVersions`「」を参照してください。

> **Note**
>
> AWS は、イメージセットに加えられたすべての変更 HealthImaging を記録します。画像セットのメタデータを更新すると、画像セット履歴に新しいバージョンが作成されます。詳細については、「画像セットメタデータの更新」を参照してください。

画像セットのバージョンを一覧表示するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   データストアの詳細ページが開き、デフォルトで [画像セット] タブが選択されます。
3. 画像セットを選択します。

   [画像セットの詳細] ページが開きます。

   画像セットのバージョンは、「画像セットの詳細」セクションに表示されます。

AWS CLI および SDKs

CLI

AWS CLI

画像セットバージョンを一覧表示するには

次の `list-image-set-versions` コード例では、画像セットのバージョン履歴を一覧表示しています。

```
aws medical-imaging list-image-set-versions \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e
```

出力:

```
{
    "imageSetPropertiesList": [
        {
            "ImageSetWorkflowStatus": "UPDATED",
            "versionId": "4",
            "updatedAt": 1680029436.304,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
            "ImageSetWorkflowStatus": "UPDATED",
```

```
            "versionId": "3",
            "updatedAt": 1680029163.325,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
            "ImageSetWorkflowStatus": "COPY_FAILED",
            "versionId": "2",
            "updatedAt": 1680027455.944,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "message": "INVALID_REQUEST:  Series of SourceImageSet and
 DestinationImageSet don't match.",
            "createdAt": 1680027126.436
        },
        {
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "versionId": "1",
            "ImageSetWorkflowStatus": "COPIED",
            "createdAt": 1680027126.436
        }
    ]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットのバージョンを一覧表示する」を参照してください。

- API の詳細については、「 コマンドリファレンスListImageSetVersions」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static List<ImageSetProperties>
listMedicalImageSetVersions(MedicalImagingClient medicalImagingClient,
            String datastoreId,
            String imagesetId) {
        try {
            ListImageSetVersionsRequest getImageSetRequest =
ListImageSetVersionsRequest.builder()
```

```
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId)
                    .build();

            ListImageSetVersionsIterable responses = medicalImagingClient
                    .listImageSetVersionsPaginator(getImageSetRequest);
            List<ImageSetProperties> imageSetProperties = new ArrayList<>();
            responses.stream().forEach(response ->
 imageSetProperties.addAll(response.imageSetPropertiesList()));

            return imageSetProperties;
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API の詳細については、「 API リファレンスListImageSetVersions」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
```

```
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- API の詳細については、「 API リファレンスListImageSetVersions」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_image_set_versions(self, datastore_id, image_set_id):
        """
        List the image set versions.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :return: The list of image set versions.
        """
        try:
            paginator = self.health_imaging_client.get_paginator(
                "list_image_set_versions"
            )
            page_iterator = paginator.paginate(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
            image_set_properties_list = []
            for page in page_iterator:
                image_set_properties_list.extend(page["imageSetPropertiesList"])
        except ClientError as err:
            logger.error(
                "Couldn't list image set versions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return image_set_properties_list
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 ListImageSetVersions AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

画像セットメタデータの更新

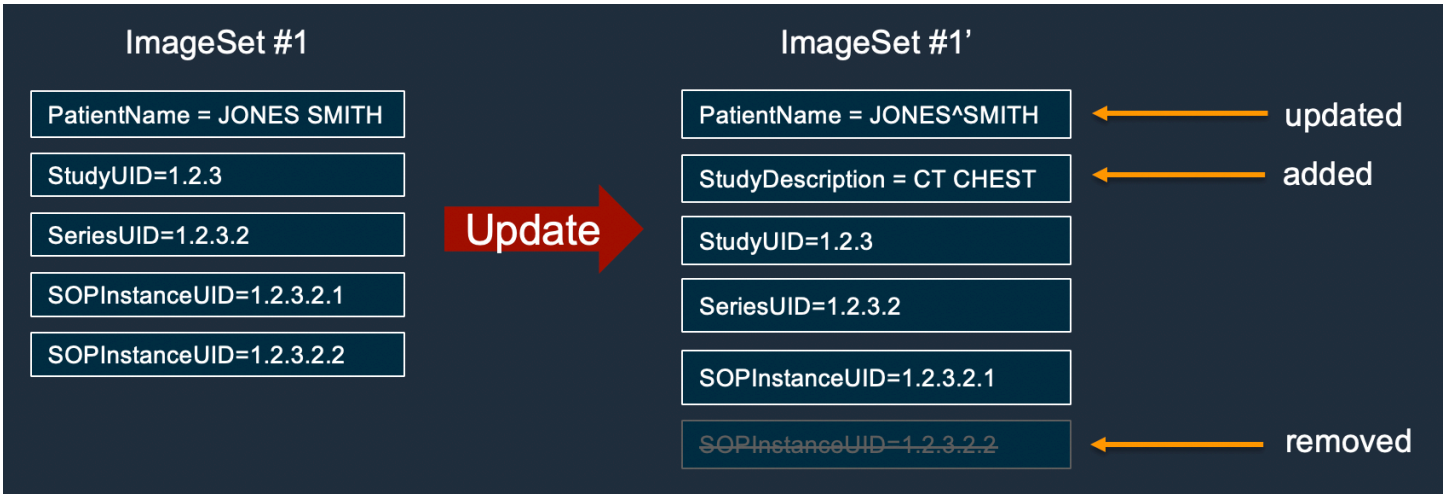
UpdateImageSetMetadata アクションを使用して、AWS で画像セットメタデータを更新します HealthImaging。この非同期プロセスを使用して、インポート中に作成される DICOM 正規化要素の兆候である画像セットメタデータ属性を追加、更新、削除できます。UpdateImageSetMetadataアクションを使用して、シリーズインスタンスとSOPインスタンスを削除し、画像セットを外部システムと同期させたり、画像セットのメタデータを匿名化したりすることもできます。詳細については、AWS HealthImaging API リファレンスのUpdateImageSetMetadata「」を参照してください。

UpdateImageSetMetadata を理解する

> **Note**
>
> 実際の DICOM インポートでは、画像セットメタデータの属性を更新、追加、削除する必要があります。画像セットのメタデータを更新するときは、次の点に注意してください。
>
> - 画像セットのメタデータを更新すると、画像セット履歴に新しいバージョンが作成されます。詳細については、「画像セットのバージョンを一覧表示する」を参照してください。

- 画像セットのメタデータの更新は非同期プロセスです。したがって、 imageSetStateおよび imageSetWorkflowStatusレスポンス要素は、ロックされた画像セットの各状態とステータスを提供するために使用できます。ロックされた画像セットには他の書き込み操作は実行できません。
- DICOM 要素の制約はメタデータの更新に適用されます。詳細については、「DICOM メタデータの制約」を参照してください。
- 画像セットメタデータの更新アクションが成功しない場合は、 を呼び出してmessageレスポンス要素を確認します。

次の図は、 で更新される画像セットメタデータを示しています HealthImaging。



画像セットのメタデータを更新するには

AWS へのアクセス設定に基づいてタブを選択します HealthImaging。

AWS CLI および SDKs

CLI

AWS CLI

画像セットメタデータに属性を挿入または更新するには

次のupdate-image-set-metadataコード例では、画像セットメタデータに属性を挿入または更新します。

```
aws medical-imaging update-image-set-metadata \
```

```
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
    "DICOMUpdates": {
        "updatableAttributes":
 "eyJTY2hlbWFWZXJzaW9uIjoxLjEsIlBhdGllbnQiOnsiRElDT00iOnsiUGF0aWVudE5hbWUiOiJNWF5NWCJ9fX0
    }
}
```

注：updatableAttributes は Base64 でエンコードされた JSON 文字列です。こちらはエンコードされていない JSON 文字列です。

{SchemaVersion"":1.1,""":{"DICOM":{"PatientName""MX^MX"}}}

出力:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

画像セットメタデータから属性を削除するには

次のupdate-image-set-metadataコード例では、画像セットメタデータから 属性を削除します。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
    "DICOMUpdates": {
        "removableAttributes":
 "e1NjaGVtYVZlcnNpb246MS4xLFN0dWR5OntESUNPTTp7U3R1ZHlEZXNjcmlwdGlvbjpDSEVTVH19fQo="
    }
}
```

注：removableAttributes は Base64 でエンコードされた JSON 文字列です。こちらはエンコードされていない JSON 文字列です。キーと値は、削除する属性と一致する必要があります。

{SchemaVersion"":1.1,"Study":{"DICOM":{"StudyDescription""CHEST"}}}

出力:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

イメージセットメタデータからインスタンスを削除するには

次のupdate-image-set-metadataコード例では、イメージセットメタデータからインスタンスを削除します。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
```

```
    "DICOMUpdates": {
        "removableAttributes":
 "eezEuMS4xLjEuMS4xLjEyMzQ1LjEyMzQ1Njc4OTAxMi4xMjMuMTIzNDU2Nzg5MDEyMzQuMTp7SW5zdGFuY2VzOn
    }
}
```

注：removableAttributes は Base64 でエンコードされた JSON 文字列です。こちらはエンコードされていない JSON 文字列です。

{"1.1.1.1.1.12345.123456789012.123.12345678901234.1":{"Instances":
{"1.1.1.1.1.1.1.123456789012.123.12345678901234.1":{}}}}}

出力:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットメタデータの更新」を参照してください。

- API の詳細については、「 コマンドリファレンスUpdateImageSetMetadata」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static void updateMedicalImageSetMetadata(MedicalImagingClient
 medicalImagingClient,
                                                      String datastoreId,
                                                      String imagesetId,
                                                      String versionId,
                                                      MetadataUpdates
 metadataUpdates) {
        try {
```

```
            UpdateImageSetMetadataRequest updateImageSetMetadataRequest =
UpdateImageSetMetadataRequest
                    .builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId)
                    .latestVersionId(versionId)
                    .updateImageSetMetadataUpdates(metadataUpdates)
                    .build();

            UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

            System.out.println("The image set metadata was updated" + response);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

ユースケース #1: 属性を挿入または更新します。

```
            final String insertAttributes = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "DICOM": {
                          "StudyDescription": "CT CHEST"
                        }
                      }
                    }
                    """;
            MetadataUpdates metadataInsertUpdates = MetadataUpdates.builder()
                    .dicomUpdates(DICOMUpdates.builder()
                            .updatableAttributes(SdkBytes.fromByteBuffer(
                                    ByteBuffer.wrap(insertAttributes
                                            .getBytes(StandardCharsets.UTF_8))))
                            .build())
                    .build();

            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                    versionid, metadataInsertUpdates);
```

ユースケース #2: 属性を削除します。

```
            final String removeAttributes = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "DICOM": {
                          "StudyDescription": "CT CHEST"
                        }
                      }
                    }
                    """;
            MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                    .dicomUpdates(DICOMUpdates.builder()
                            .removableAttributes(SdkBytes.fromByteBuffer(
                                    ByteBuffer.wrap(removeAttributes
                                            .getBytes(StandardCharsets.UTF_8))))
                            .build())
                    .build();

            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                    versionid, metadataRemoveUpdates);
```

ユースケース #3: インスタンスを削除します。

```
            final String removeInstance = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "Series": {
                          "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1":
{
                            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
                          }
                        }
                      }
```

```
                    }
                    """;
            MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                    .dicomUpdates(DICOMUpdates.builder()
                            .removableAttributes(SdkBytes.fromByteBuffer(
                                    ByteBuffer.wrap(removeInstance
                                            .getBytes(StandardCharsets.UTF_8))))
                            .build())
                    .build();

            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
 imagesetId,
                    versionid, metadataRemoveUpdates);
```

- API の詳細については、「 API リファレンスUpdateImageSetMetadata」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
```

```
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
};
```

ユースケース #1: 属性を挿入または更新します。

```
        const insertAttributes =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
```

```
            "DICOMUpdates": {
                "updatableAttributes":
                    new TextEncoder().encode(insertAttributes)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

ユースケース #2: 属性を削除します。

```
        // Attribute key and value must match the existing attribute.
        const remove_attribute =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "removableAttributes":
                    new TextEncoder().encode(remove_attribute)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

ユースケース #3: インスタンスを削除します。

```
        const remove_instance =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "Series": {
                        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                            "Instances": {
```

```
 "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
                        }
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "removableAttributes":
                    new TextEncoder().encode(remove_instance)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

- API の詳細については、「 API リファレンスUpdateImageSetMetadata」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def update_image_set_metadata(
        self, datastore_id, image_set_id, version_id, metadata
    ):
        """
        Update the metadata of an image set.
```

```
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param metadata: The image set metadata as a dictionary.
            For example {"DICOMUpdates": {"updatableAttributes":
            "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{\"PatientName\":
\"Garcia^Gloria\"}}}"}}
        :return: The updated image set metadata.
        """
        try:
            updated_metadata =
 self.health_imaging_client.update_image_set_metadata(
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                latestVersionId=version_id,
                updateImageSetMetadataUpdates=metadata,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update image set metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return updated_metadata
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

ユースケース #1: 属性を挿入または更新します。

```
            attributes = """{
                    "SchemaVersion": 1.1,
                    "Study": {
                        "DICOM": {
                            "StudyDescription": "CT CHEST"
```

```
                    }
                }
            }"""
        metadata = {"DICOMUpdates": {"updatableAttributes": attributes}}

        self.update_image_set_metadata(
            data_store_id, image_set_id, version_id, metadata
        )
```

ユースケース #2: 属性を削除します。

```
        # Attribute key and value must match the existing attribute.
        attributes = """{
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            }"""
        metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

        self.update_image_set_metadata(
            data_store_id, image_set_id, version_id, metadata
        )
```

ユースケース #3: インスタンスを削除します。

```
        attributes = """{
                "SchemaVersion": 1.1,
                "Study": {
                    "Series": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
                        }
                    }
                }
```

```
            }"""
        metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

        self.update_image_set_metadata(
            data_store_id, image_set_id, version_id, metadata
        )
```

- API の詳細については、 UpdateImageSetMetadata AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

画像セットのコピー

CopyImageSet アクションを使用して、 で画像セットをコピーします HealthImaging。この非同期プロセスを使用して、画像セットの内容を新規または既存の画像セットにコピーします。新しいイメージにコピーして画像セットを分割したり、別のコピーを作成したりできます。既存の画像セットにコピーして 2 つの画像セットを結合することもできます。詳細については、AWS HealthImaging API リファレンスのCopyImageSet「」を参照してください。

CopyImageSet を理解する

> **Note**
>
> 画像セットをコピーするときは、次の点に注意してください。
>
> - 画像セットをコピーすると、画像セットの履歴に新しいバージョンが作成されます。詳細については、「画像セットのバージョンを一覧表示する」を参照してください。
> - 画像セットのコピーは非同期プロセスです。したがって、状態 (imageSetState) とステータス (imageSetWorkflowStatus) のレスポンス要素は、ロックされた画像セットでどのようなオペレーションが実行されているかを知らせるために使用できます。ロックされた画像セットでは、他の書き込みオペレーションを実行できません。

- CopyImageSet を成功させるには、一意の SOP インスタンス UIDsが必要です。したがって、不要な画像セットから削除して、正しい SOP インスタンスを選択する必要があります。
- 画像セットコピーのアクションが成功しなかった場合は、GetImageSet を呼び出して、プロパティ message を確認してください。詳細については、「画像セットのプロパティの取得」を参照してください。
- 実際に DICOM をインポートすると、DICOM シリーズごとに複数の画像セットが作成される可能性があります。CopyImageSet アクションを使用する際には、以下の点を考慮してください。
  - ある画像セットから別の画像セットにインスタンスをコピーする
  - コピーでは、両方の画像セットに一貫したメタデータが必要です

画像セットをコピーするには

AWS へのアクセス設定に基づいてタブを選択します HealthImaging。

AWS CLI および SDKs

CLI

AWS CLI

例 1 : コピー先を指定せずに画像セットをコピーするには。

次の copy-image-set コード例では、コピー先を指定せずに画像セットの複製コピーを作成します。

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "1" } }'
```

出力:

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
```

```
        "updatedAt": 1680042357.432,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042357.432,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
}
```

例 2 : コピー先を指定して画像セットをコピーするには。

次の copy-image-set コード例では、コピー先を指定して画像セットの複製コピーを作成します。

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "1" },
 "destinationImageSet": { "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
 "latestVersionId": "1"} }'
```

出力:

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042505.135,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042505.135,
```

```
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットのコピー」を参照してください。

- API の詳細については、「 コマンドリファレンスCopyImageSet」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static String copyMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imageSetId,
           String latestVersionId,
           String destinationImageSetId,
           String destinationVersionId) {

       try {
           CopySourceImageSetInformation copySourceImageSetInformation =
CopySourceImageSetInformation.builder()
                   .latestVersionId(latestVersionId)
                   .build();

           CopyImageSetInformation.Builder copyImageSetBuilder =
CopyImageSetInformation.builder()
                   .sourceImageSet(copySourceImageSetInformation);

           if (destinationImageSetId != null) {
               copyImageSetBuilder =
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()
                       .imageSetId(destinationImageSetId)
                       .latestVersionId(destinationVersionId)
                       .build());
           }
```

```
            CopyImageSetRequest copyImageSetRequest =
CopyImageSetRequest.builder()
                    .datastoreId(datastoreId)
                    .sourceImageSetId(imageSetId)
                    .copyImageSetInformation(copyImageSetBuilder.build())
                    .build();

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

            return response.destinationImageSetProperties().imageSetId();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return "";
    }
```

- API の詳細については、「 API リファレンスCopyImageSet」の「」を参照してください。AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

イメージセットをコピーするためのユーティリティ関数。

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
```

```
 * @param {string} destinationImageSetId - The optional ID of the destination
 image set.
 * @param {string} destinationVersionId - The optional version ID of the
 destination image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //      },
  //       datastoreId: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
```

```
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

コピー先を指定せずにイメージセットをコピーします。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

コピー先を指定してイメージセットをコピーします。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
```

```
    );
  } catch (err) {
    console.error(err);
  }
```

- API の詳細については、「 API リファレンスCopyImageSet」の「」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

イメージセットをコピーするためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def copy_image_set(
        self,
        datastore_id,
        image_set_id,
        version_id,
        destination_image_set_id=None,
        destination_version_id=None,
    ):
        """
        Copy an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param destination_image_set_id: The ID of the optional destination image
 set.
```

```
        :param destination_version_id: The ID of the optional destination image
set version.
        :return: The copied image set ID.
        """
        try:
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }
            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
                    "imageSetId": destination_image_set_id,
                    "latestVersionId": destination_version_id,
                }
            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
        except ClientError as err:
            logger.error(
                "Couldn't copy image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return copy_results["destinationImageSetProperties"]["imageSetId"]
```

コピー先を指定せずにイメージセットをコピーします。

```
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }

            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
```

コピー先を指定してイメージセットをコピーします。

```
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }

            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
                    "imageSetId": destination_image_set_id,
                    "latestVersionId": destination_version_id,
                }

            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 CopyImageSet AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

画像セットの削除

DeleteImageSet アクションを使用して、 のイメージセットを削除します HealthImaging。次のメニューでは、 の手順と、 AWS Management Console および AWS CLI AWS SDKs。詳細については、AWS HealthImaging API リファレンスのDeleteImageSet「」を参照してください。

画像セットを削除するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   データストアの詳細ページが開き、デフォルトで [画像セット] タブが選択されます。
3. 画像セットを選択し、[削除] を選択します。

   [画像セットを削除] モーダルが開きます。
4. 画像セットの ID を入力し、画像セットを削除を選択します。

AWS CLI および SDKs

C++

SDK for C++

```
//! Routine which deletes an AWS HealthImaging image set.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The image set ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::deleteImageSet(
        const Aws::String &dataStoreID, const Aws::String &imageSetID,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::DeleteImageSetRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    Aws::MedicalImaging::Model::DeleteImageSetOutcome outcome =
 client.DeleteImageSet(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted image set " << imageSetID
                  << " from data store " << dataStoreID << std::endl;
    }
    else {
```

```
        std::cerr << "Error deleting image set " << imageSetID << " from data
 store "
                  << dataStoreID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「 API リファレンスDeleteImageSet」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

画像セットを削除するには

以下の `delete-image-set` コード例は画像セットを削除しています。

```
aws medical-imaging delete-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e
```

出力:

```
{
    "imageSetWorkflowStatus": "DELETING",
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットの削除」を参照してください。

- API の詳細については、「 コマンドリファレンスDeleteImageSet」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imagesetId) {
       try {
           DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId)
                   .build();

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

           System.out.println("The image set was deleted.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「 API リファレンスDeleteImageSet」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- API の詳細については、「 API リファレンスDeleteImageSet」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def delete_image_set(self, datastore_id, image_set_id):
        """
        Delete an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :return: The delete results.
        """
        try:
            delete_results = self.health_imaging_client.delete_image_set(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return delete_results
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
```

```
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 DeleteImageSet AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS によるリソースのタグ付け HealthImaging

AWS HealthImaging リソース (データストアと画像セット) にメタデータをタグ形式で割り当てることができます。各タグは、ユーザー定義のキーと値で構成されるラベルです。タグは、リソースの管理、識別、整理、検索、フィルタリングに役立ちます。

> [重要]
>
> 保健情報 (PHI)、個人を特定できる情報 (PII)などの機密情報や秘匿性の高い情報はタグに格納しないでください。タグは、プライベートデータや機密データに使用することを意図していません。

以下のトピックでは、 AWS Management Console、 AWS CLI、 AWS SDKs を使用して HealthImaging タグ付けオペレーションを使用する方法について説明します。詳細については、「 AWS 全般のリファレンス ガイド」の「 AWS リソースのタグ付け」を参照してください。

トピック

- リソースのタギング
- リソースのタグを一覧表示します
- リソースのタグを削除します

リソースのタギング

`TagResource` アクションを使用して、AWS のリソースにタグを付けます HealthImaging。次のコード例は、、 AWS Management Console AWS CLI、 AWS SDKs で `TagResource`アクションを使用する方法を示しています。詳細については、「 AWS 全般のリファレンス ガイド」の「 AWS リソースのタグ付け」を参照してください。

リソース (データストア) にタグを付けるには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。

2. データストアを選択します。

   [データストアの詳細] ページが開きます。
3. 詳細 タブを選択します。
4. [タグ] セクションで、[タグの管理] を選択します。

   [タグの管理] ページが開きます。
5. [新しいタグを追加] をクリックします。
6. [キー] を入力し、オプションで [値] を入力します。
7. [変更の保存] を選択します。

AWS CLI および SDKs

CLI

AWS CLI

例 1 : データストアにタグを付けるには

次の `tag-resource` コード例では、データストアにタグを付けています。

```
aws medical-imaging tag-resource \
  --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
  --tags '{"Deployment":"Development"}'
```

このコマンドでは何も出力されません。

例 2 : 画像セットにタグを付けるには

次の `tag-resource` コード例では、画像セットにタグを付けています。

```
aws medical-imaging tag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
    --tags '{"Deployment":"Development"}'
```

このコマンドでは何も出力されません。

詳細については、「 AWS HealthImaging デベロッパーガイド」の「 によるリソースのタグ付け AWS HealthImaging」を参照してください。

- API の詳細については、「 コマンドリファレンスTagResource」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void tagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Map<String, String> tags) {
       try {
           TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tags(tags)
                   .build();

           medicalImagingClient.tagResource(tagResourceRequest);

           System.out.println("Tags have been added to the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「 API リファレンスTagResource」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスTagResource」の「」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def tag_resource(self, resource_arn, tags):
        """
        Tag a resource.

        :param resource_arn: The ARN of the resource.
        :param tags: The tags to apply.
        """
        try:
            self.health_imaging_client.tag_resource(resourceArn=resource_arn,
 tags=tags)
        except ClientError as err:
            logger.error(
                "Couldn't tag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 TagResource AWS SDK for Python (Boto3) API リファレンス の「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

リソースのタグを一覧表示します

ListTagsForResource アクションを使用して、AWS のリソースのタグを一覧表示します HealthImaging。次のコード例は、 AWS Management Console、 AWS CLI、 AWS SDKs で ListTagsForResourceアクションを使用する方法を示しています。詳細については、「 AWS 全般のリファレンス ガイド」の「 AWS リソースのタグ付け」を参照してください。

リソース (データストア) のタグを一覧表示するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   [データストアの詳細] ページが開きます。
3. 詳細 タブを選択します。

   [タグ] セクションに、すべてのデータストアタグが一覧表示されます。

AWS CLI および SDKs

CLI

AWS CLI

例 1 : データストアリソースのタグを一覧表示するには

次の list-tags-for-resource コード例では、データストアのタグを一覧表示しています。

```
aws medical-imaging list-tags-for-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"
```

出力:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

例 2 : 画像セットリソースのタグを一覧表示するには

次の `list-tags-for-resource` コード例では、画像セットのタグを一覧表示しています。

```
aws medical-imaging list-tags-for-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b"
```

出力:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「 によるリソースのタグ付け AWS HealthImaging」を参照してください。

- API の詳細については、「 コマンドリファレンスListTagsForResource」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static ListTagsForResourceResponse
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
```

```
            String resourceArn) {
        try {
            ListTagsForResourceRequest listTagsForResourceRequest =
 ListTagsForResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .build();

            return
 medicalImagingClient.listTagsForResource(listTagsForResourceRequest);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API の詳細については、「 API リファレンスListTagsForResource」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi"
) => {
```

```
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスListTagsForResource」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_tags_for_resource(self, resource_arn):
        """
        List the tags for a resource.
```

```
        :param resource_arn: The ARN of the resource.
        :return: The list of tags.
        """
        try:
            tags = self.health_imaging_client.list_tags_for_resource(
                resourceArn=resource_arn
            )
        except ClientError as err:
            logger.error(
                "Couldn't list tags for resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tags["tags"]
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
client = boto3.client("medical-imaging")
medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 ListTagsForResource AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

リソースのタグを削除します

UntagResource アクションを使用して、AWS のリソースのタグを解除します HealthImaging。次のコード例は、 AWS Management Console、 AWS CLI、 AWS SDKs で UntagResourceアクションを使用する方法を示しています。詳細については、「 AWS 全般のリファレンス ガイド」の「 AWS リソースのタグ付け」を参照してください。

リソース (データストア) のタグを削除するには

AWS へのアクセス設定に基づいてメニューを選択します HealthImaging。

AWS コンソール

1. HealthImaging コンソールのデータストアページを開きます。
2. データストアを選択します。

   [データストアの詳細] ページが開きます。
3. 詳細 タブを選択します。
4. [タグ] セクションで、[タグの管理] を選択します。

   [タグの管理] ページが開きます。
5. 削除するタグの横にある [削除] を選択します。
6. [変更の保存] を選択します。

AWS CLI および SDKs

CLI

AWS CLI

例 1 : データストアのタグを削除するには

次の untag-resource コード例では、データストアにタグを削除します。

```
aws medical-imaging untag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
    --tag-keys '["Deployment"]'
```

このコマンドでは何も出力されません。

例 2 : 画像セットにタグを削除するには

次の untag-resource コード例では、画像セットにタグを削除します。

```
aws medical-imaging untag-resource \
```

```
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
    --tag-keys '["Deployment"]'
```

このコマンドでは何も出力されません。

詳細については、「 AWS HealthImaging デベロッパーガイド」の「 によるリソースのタグ付け AWS HealthImaging」を参照してください。

- API の詳細については、「 コマンドリファレンスUntagResource」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void untagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
            String resourceArn,
            Collection<String> tagKeys) {
        try {
            UntagResourceRequest untagResourceRequest =
UntagResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .tagKeys(tagKeys)
                    .build();

            medicalImagingClient.untagResource(untagResourceRequest);

            System.out.println("Tags have been removed from the resource.");
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「 API リファレンスUntagResource」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスUntagResource」の「」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def untag_resource(self, resource_arn, tag_keys):
        """
        Untag a resource.

        :param resource_arn: The ARN of the resource.
        :param tag_keys: The tag keys to remove.
        """
        try:
            self.health_imaging_client.untag_resource(
                resourceArn=resource_arn, tagKeys=tag_keys
            )
        except ClientError as err:
            logger.error(
                "Couldn't untag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
```

```
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 UntagResource AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDKs HealthImaging を使用するためのコード例

次のコード例は、 Software AWS Development Kit (SDK) HealthImaging で を使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

AWS SDK デベロッパー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 HealthImaging での 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。

開始方法

こんにち HealthImagingは

次のコード例は、 の使用を開始する方法を示しています HealthImaging。

C++

SDK for C++

C MakeLists.txt CMake ファイルのコード。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS medical-imaging)

# Set this project's name.
project("hello_health-imaging")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)
```

```
# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
    # and set the proper subdirectory to the executable location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_health_imaging.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_health_imaging.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/medical-imaging/MedicalImagingClient.h>
#include <aws/medical-imaging/model/ListDatastoresRequest.h>

#include <iostream>

/*
 *  A "Hello HealthImaging" starter application which initializes an AWS
 HealthImaging (HealthImaging) client
```

```
 *  and lists the HealthImaging data stores in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_health-imaging'
 *
 */
#include <aws/core/auth/AWSCredentialsProviderChain.h>
#include <aws/core/platform/Environment.h>

int main(int argc, char **argv) {
    (void) argc;
    (void) argv;
    Aws::SDKOptions options;
    //   Optional: change the log level for debugging.
    //   options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;

    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::MedicalImaging::MedicalImagingClient
 medicalImagingClient(clientConfig);
        Aws::MedicalImaging::Model::ListDatastoresRequest listDatastoresRequest;

        Aws::Vector<Aws::MedicalImaging::Model::DatastoreSummary>
 allDataStoreSummaries;
        Aws::String nextToken; // Used for paginated results.
        do {
            if (!nextToken.empty()) {
                listDatastoresRequest.SetNextToken(nextToken);
            }
            Aws::MedicalImaging::Model::ListDatastoresOutcome
 listDatastoresOutcome =
                    medicalImagingClient.ListDatastores(listDatastoresRequest);
            if (listDatastoresOutcome.IsSuccess()) {
                const Aws::Vector<Aws::MedicalImaging::Model::DatastoreSummary>
 &dataStoreSummaries =

 listDatastoresOutcome.GetResult().GetDatastoreSummaries();
                allDataStoreSummaries.insert(allDataStoreSummaries.cend(),
                                             dataStoreSummaries.cbegin(),
```

```
                                                dataStoreSummaries.cend());
                nextToken = listDatastoresOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "ListDatastores error: "
                          << listDatastoresOutcome.GetError().GetMessage() <<
 std::endl;
                break;
            }
        } while (!nextToken.empty());

        std::cout << allDataStoreSummaries.size() << " HealthImaging data "
                  << ((allDataStoreSummaries.size() == 1) ?
                      "store was retrieved." : "stores were retrieved.") <<
 std::endl;

        for (auto const &dataStoreSummary: allDataStoreSummaries) {
            std::cout << "  Datastore: " << dataStoreSummary.GetDatastoreName()
                      << std::endl;
            std::cout << "  Datastore ID: " << dataStoreSummary.GetDatastoreId()
                      << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- API の詳細については、「 API リファレンスListDatastores」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- API の詳細については、「 API リファレンスListDatastores」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
```

```
def hello_medical_imaging(medical_imaging_client):
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon HealthImaging
    client and list the data stores in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param medical_imaging_client: A Boto3 Amazon HealthImaging Client object.
    """
    print("Hello, Amazon Health Imaging! Let's list some of your data stores:\n")
    try:
        paginator = medical_imaging_client.get_paginator("list_datastores")
        page_iterator = paginator.paginate()
        datastore_summaries = []
        for page in page_iterator:
            datastore_summaries.extend(page["datastoreSummaries"])
        print("\tData Stores:")
        for ds in datastore_summaries:
            print(f"\t\tDatastore: {ds['datastoreName']} ID {ds['datastoreId']}")
    except ClientError as err:
        logger.error(
            "Couldn't list data stores. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise


if __name__ == "__main__":
    hello_medical_imaging(boto3.client("medical-imaging"))
```

- API の詳細については、 ListDatastores AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コードの例

AWS SDKs HealthImaging を使用するためのアクション

次のコード例は、 AWS SDKsで個々の HealthImagingアクションを実行する方法を示しています。これらの抜粋は HealthImaging API を呼び出し、コンテキスト内で実行する必要がある大規模なプログラムからのコードの抜粋です。各例には GitHub、コードの設定と実行の手順を示す へのリンクが含まれています。

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。詳細な一覧については、「AWS HealthImaging API リファレンス」を参照してください。

例

- AWS SDK または CLI CopyImageSetで を使用する
- AWS SDK または CLI CreateDatastoreで を使用する
- AWS SDK または CLI DeleteDatastoreで を使用する
- AWS SDK または CLI DeleteImageSetで を使用する
- AWS SDK または CLI GetDICOMImportJobで を使用する
- AWS SDK または CLI GetDatastoreで を使用する
- AWS SDK または CLI GetImageFrameで を使用する
- AWS SDK または CLI GetImageSetで を使用する
- AWS SDK または CLI GetImageSetMetadataで を使用する
- AWS SDK または CLI ListDICOMImportJobsで を使用する
- AWS SDK または CLI ListDatastoresで を使用する
- AWS SDK または CLI ListImageSetVersionsで を使用する
- AWS SDK または CLI ListTagsForResourceで を使用する
- AWS SDK または CLI SearchImageSetsで を使用する
- AWS SDK または CLI StartDICOMImportJobで を使用する
- AWS SDK または CLI TagResourceで を使用する
- AWS SDK または CLI UntagResourceで を使用する
- AWS SDK または CLI UpdateImageSetMetadataで を使用する

AWS SDK または CLI **CopyImageSet**で を使用する

以下のコード例は、CopyImageSet の使用方法を示しています。

CLI

AWS CLI

例 1：コピー先を指定せずに画像セットをコピーするには。

次の copy-image-set コード例では、コピー先を指定せずに画像セットの複製コピーを作成します。

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "1" } }'
```

出力:

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042357.432,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042357.432,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
}
```

例 2 : コピー先を指定して画像セットをコピーするには。

次の copy-image-set コード例では、コピー先を指定して画像セットの複製コピーを作成します。

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "1" },
 "destinationImageSet": { "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
 "latestVersionId": "1"} }'
```

出力:

```
{
```

```
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042505.135,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042505.135,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットのコピー」を参照してください。

- API の詳細については、「 コマンドリファレンスCopyImageSet」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static String copyMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imageSetId,
           String latestVersionId,
           String destinationImageSetId,
           String destinationVersionId) {

       try {
           CopySourceImageSetInformation copySourceImageSetInformation =
CopySourceImageSetInformation.builder()
                   .latestVersionId(latestVersionId)
                   .build();
```

```
            CopyImageSetInformation.Builder copyImageSetBuilder =
CopyImageSetInformation.builder()
                    .sourceImageSet(copySourceImageSetInformation);

            if (destinationImageSetId != null) {
                copyImageSetBuilder =
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()
                        .imageSetId(destinationImageSetId)
                        .latestVersionId(destinationVersionId)
                        .build());
            }

            CopyImageSetRequest copyImageSetRequest =
CopyImageSetRequest.builder()
                    .datastoreId(datastoreId)
                    .sourceImageSetId(imageSetId)
                    .copyImageSetInformation(copyImageSetBuilder.build())
                    .build();

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

            return response.destinationImageSetProperties().imageSetId();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return "";
    }
```

- API の詳細については、「 API リファレンスCopyImageSet」の「」を参照してください。AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

イメージセットをコピーするためのユーティリティ関数。

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination
 image set.
 * @param {string} destinationVersionId - The optional version ID of the
 destination image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
```

```
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //      },
  //       datastoreId: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

コピー先を指定せずにイメージセットをコピーします。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
```

```
    console.error(err);
  }
```

コピー先を指定してイメージセットをコピーします。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

- API の詳細については、「 API リファレンスCopyImageSet」の「」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

イメージセットをコピーするためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def copy_image_set(
        self,
        datastore_id,
```

```
        image_set_id,
        version_id,
        destination_image_set_id=None,
        destination_version_id=None,
    ):
        """
        Copy an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param destination_image_set_id: The ID of the optional destination image
set.
        :param destination_version_id: The ID of the optional destination image
set version.
        :return: The copied image set ID.
        """
        try:
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }
            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
                    "imageSetId": destination_image_set_id,
                    "latestVersionId": destination_version_id,
                }
            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
        except ClientError as err:
            logger.error(
                "Couldn't copy image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return copy_results["destinationImageSetProperties"]["imageSetId"]
```

コピー先を指定せずにイメージセットをコピーします。

```
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }

            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
```

コピー先を指定してイメージセットをコピーします。

```
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }

            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
                    "imageSetId": destination_image_set_id,
                    "latestVersionId": destination_version_id,
                }

            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 CopyImageSet AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **CreateDatastore**で を使用する

以下のコード例は、CreateDatastore の使用方法を示しています。

Bash

AWS CLI Bash スクリプトを使用する

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function imaging_create_datastore
#
# This function creates an AWS HealthImaging data store for importing DICOM P10
 files.
#
# Parameters:
#       -n data_store_name - The name of the data store.
#
# Returns:
#       The datastore ID.
#    And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
```

```
function imaging_create_datastore() {
  local datastore_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_create_datastore"
    echo "Creates an AWS HealthImaging data store for importing DICOM P10 files."
    echo "  -n data_store_name - The name of the data store."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) datastore_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$datastore_name" ]]; then
    errecho "ERROR: You must provide a data store name with the -n parameter."
    usage
    return 1
  fi

  response=$(aws medical-imaging create-datastore \
    --datastore-name "$datastore_name" \
    --output text \
    --query 'datastoreId')

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports medical-imaging create-datastore operation
 failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- API の詳細については、「 コマンドリファレンスCreateDatastore」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアを作成するには

次の create-datastore コード例では、my-datastore という名が付けられたデータストアを作成しています。

```
aws medical-imaging create-datastore \
    --datastore-name "my-datastore"
```

出力:

```
{
    "datastoreId": "12345678901234567890123456789012",
    "datastoreStatus": "CREATING"
}
```

詳細については、「 デベロッパーガイド」の「データストアの作成」を参照してください。 AWS HealthImaging

- API の詳細については、「 コマンドリファレンスCreateDatastore」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static String createMedicalImageDatastore(MedicalImagingClient
medicalImagingClient,
           String datastoreName) {
       try {
           CreateDatastoreRequest datastoreRequest =
CreateDatastoreRequest.builder()
                   .datastoreName(datastoreName)
                   .build();
           CreateDatastoreResponse response =
medicalImagingClient.createDatastore(datastoreRequest);
           return response.datastoreId();
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return "";
   }
```

- API の詳細については、「 API リファレンスCreateDatastore」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
```

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- API の詳細については、「 API リファレンスCreateDatastore」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client
```

```
    def create_datastore(self, name):
        """
        Create a data store.

        :param name: The name of the data store to create.
        :return: The data store ID.
        """
        try:
            data_store =
self.health_imaging_client.create_datastore(datastoreName=name)
        except ClientError as err:
            logger.error(
                "Couldn't create data store %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return data_store["datastoreId"]
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 CreateDatastore AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **DeleteDatastore**で を使用する

以下のコード例は、DeleteDatastore の使用方法を示しています。

Bash

AWS CLI Bash スクリプトを使用する

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function imaging_delete_datastore
#
# This function deletes an AWS HealthImaging data store.
#
# Parameters:
#       -i datastore_id - The ID of the data store.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function imaging_delete_datastore() {
  local datastore_id response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_delete_datastore"
    echo "Deletes an AWS HealthImaging data store."
    echo "  -i datastore_id - The ID of the data store."
    echo ""
```

```
  }

  # Retrieve the calling parameters.
  while getopts "i:h" option; do
    case "${option}" in
      i) datastore_id="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$datastore_id" ]]; then
    errecho "ERROR: You must provide a data store ID with the -i parameter."
    usage
    return 1
  fi

  response=$(aws medical-imaging delete-datastore \
    --datastore-id "$datastore_id")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports medical-imaging delete-datastore operation
 failed.$response"
    return 1
  fi

  return 0
}
```

- API の詳細については、「 コマンドリファレンスDeleteDatastore」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアを削除するには

次の delete-datastore コード例では、データストアを削除しています。

```
aws medical-imaging delete-datastore \
    --datastore-id "12345678901234567890123456789012"
```

出力:

```
{
    "datastoreId": "12345678901234567890123456789012",
    "datastoreStatus": "DELETING"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「データストアの削除」を参照してください。

- API の詳細については、「 コマンドリファレンスDeleteDatastore」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void deleteMedicalImagingDatastore(MedicalImagingClient
medicalImagingClient,
           String datastoreID) {
       try {
           DeleteDatastoreRequest datastoreRequest =
DeleteDatastoreRequest.builder()
                   .datastoreId(datastoreID)
```

```
                    .build();
            medicalImagingClient.deleteDatastore(datastoreRequest);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「 API リファレンスDeleteDatastore」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //       },
```

```
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスDeleteDatastore」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def delete_datastore(self, datastore_id):
        """
        Delete a data store.

        :param datastore_id: The ID of the data store.
        """
        try:
            self.health_imaging_client.delete_datastore(datastoreId=datastore_id)
        except ClientError as err:
            logger.error(
                "Couldn't delete data store %s. Here's why: %s: %s",
                datastore_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 DeleteDatastore AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **DeleteImageSet**で を使用する

以下のコード例は、DeleteImageSet の使用方法を示しています。

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。

- 画像セットと画像フレームを使い始めます

C++

SDK for C++

```
//! Routine which deletes an AWS HealthImaging image set.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The image set ID.
  \param clientConfig: Aws client configuration.
```

```
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::deleteImageSet(
        const Aws::String &dataStoreID, const Aws::String &imageSetID,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::DeleteImageSetRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    Aws::MedicalImaging::Model::DeleteImageSetOutcome outcome =
 client.DeleteImageSet(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted image set " << imageSetID
                  << " from data store " << dataStoreID << std::endl;
    }
    else {
        std::cerr << "Error deleting image set " << imageSetID << " from data
 store "
                  << dataStoreID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「 API リファレンスDeleteImageSet」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

画像セットを削除するには

以下の `delete-image-set` コード例は画像セットを削除しています。

```
aws medical-imaging delete-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e
```

出力:

```
{
    "imageSetWorkflowStatus": "DELETING",
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットの削除」を参照してください。

- API の詳細については、「 コマンドリファレンスDeleteImageSet」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imagesetId) {
       try {
           DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId)
                   .build();

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

           System.out.println("The image set was deleted.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「 API リファレンスDeleteImageSet」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
```

```
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- API の詳細については、「 API リファレンスDeleteImageSet」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def delete_image_set(self, datastore_id, image_set_id):
        """
        Delete an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :return: The delete results.
        """
        try:
            delete_results = self.health_imaging_client.delete_image_set(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete image set. Here's why: %s: %s",
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return delete_results
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 DeleteImageSet AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **GetDICOMImportJob**で を使用する

以下のコード例は、GetDICOMImportJob の使用方法を示しています。

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。

- 画像セットと画像フレームを使い始めます

C++

SDK for C++

```
//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
  \param clientConfig: Aws client configuration.
  \return GetDICOMImportJobOutcome: The import job outcome.
*/
Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
AwsDoc::Medical_Imaging::getDICOMImportJob(const Aws::String &dataStoreID,
                                           const Aws::String &importJobID,
                                           const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetDICOMImportJobRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetJobId(importJobID);
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome outcome =
 client.GetDICOMImportJob(
            request);
    if (!outcome.IsSuccess()) {
        std::cerr << "GetDICOMImportJob error: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome;
}
```

- API の詳細については、「 API リファレンスGetDICOMImportJob」を参照してください。AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

DICOM インポートジョブのプロパティを取得するには

次の get-dicom-import-job コード例では、DICOM インポートジョブのプロパティを取得しています。

```
aws medical-imaging get-dicom-import-job \
    --datastore-id "12345678901234567890123456789012" \
    --job-id "09876543210987654321098765432109"
```

出力:

```
{
    "jobProperties": {
        "jobId": "09876543210987654321098765432109",
        "jobName": "my-job",
        "jobStatus": "COMPLETED",
        "datastoreId": "12345678901234567890123456789012",
        "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
        "endedAt": "2022-08-12T11:29:42.285000+00:00",
        "submittedAt": "2022-08-12T11:28:11.152000+00:00",
        "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",
        "outputS3Uri": "s3://medical-imaging-output/
job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/"
    }
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「インポートジョブのプロパティの取得」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GetDICOMImportJob」を参照してください。

Java

SDK for Java 2.x

```
   public static DICOMImportJobProperties getDicomImportJob(MedicalImagingClient
medicalImagingClient,
            String datastoreId,
            String jobId) {

        try {
            GetDicomImportJobRequest getDicomImportJobRequest =
GetDicomImportJobRequest.builder()
                    .datastoreId(datastoreId)
                    .jobId(jobId)
                    .build();
            GetDicomImportJobResponse response =
medicalImagingClient.getDICOMImportJob(getDicomImportJobRequest);
            return response.jobProperties();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
   }
```

- API の詳細については、「 API リファレンスGetDICOMImportJob」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
```

```
/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスGetDICOMImportJob」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def get_dicom_import_job(self, datastore_id, job_id):
        """
        Get the properties of a DICOM import job.

        :param datastore_id: The ID of the data store.
        :param job_id: The ID of the job.
        :return: The job properties.
        """
        try:
            job = self.health_imaging_client.get_dicom_import_job(
                jobId=job_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't get DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobProperties"]
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
```

```
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 AWS SDK for GetDICOMImportJob」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **GetDatastore**で を使用する

以下のコード例は、GetDatastore の使用方法を示しています。

Bash

AWS CLI Bash スクリプトを使用する

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function imaging_get_datastore
#
# Get a data store's properties.
#
# Parameters:
#       -i data_store_id - The ID of the data store.
#
# Returns:
```

```
#       [datastore_name, datastore_id, datastore_status, datastore_arn,
 created_at, updated_at]
#    And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function imaging_get_datastore() {
  local datastore_id option OPTARG # Required to use getopts command in a
 function.
  local error_code
  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_get_datastore"
    echo "Gets a data store's properties."
    echo "  -i datastore_id - The ID of the data store."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "i:h" option; do
    case "${option}" in
      i) datastore_id="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$datastore_id" ]]; then
    errecho "ERROR: You must provide a data store ID with the -i parameter."
    usage
    return 1
  fi

  local response

  response=$(
```

```
    aws medical-imaging get-datastore \
      --datastore-id "$datastore_id" \
      --output text \
      --query "[ datastoreProperties.datastoreName,
 datastoreProperties.datastoreId, datastoreProperties.datastoreStatus,
 datastoreProperties.datastoreArn,  datastoreProperties.createdAt,
 datastoreProperties.updatedAt]"
  )
  error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-datastores operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- API の詳細については、「 コマンドリファレンスGetDatastore」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアのプロパティを取得するには

次の `get-datastore` コード例では、データストアのプロパティを取得しています。

```
aws medical-imaging get-datastore \
    --datastore-id 12345678901234567890123456789012
```

出力:

```
{
    "datastoreProperties": {
        "datastoreId": "12345678901234567890123456789012",
        "datastoreName": "TestDatastore123",
        "datastoreStatus": "ACTIVE",
        "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
        "createdAt": "2022-11-15T23:33:09.643000+00:00",
        "updatedAt": "2022-11-15T23:33:09.643000+00:00"
    }
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「データストアプロパティの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetDatastore」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static DatastoreProperties
 getMedicalImageDatastore(MedicalImagingClient medicalImagingClient,
            String datastoreID) {
        try {
            GetDatastoreRequest datastoreRequest = GetDatastoreRequest.builder()
                    .datastoreId(datastoreID)
                    .build();
            GetDatastoreResponse response =
 medicalImagingClient.getDatastore(datastoreRequest);
            return response.datastoreProperties();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API の詳細については、「 API リファレンスGetDatastore」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxx:datastore/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
```

```
  // }
  return response["datastoreProperties"];
};
```

- API の詳細については、「 API リファレンスGetDatastore」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def get_datastore_properties(self, datastore_id):
        """
        Get the properties of a data store.

        :param datastore_id: The ID of the data store.
        :return: The data store properties.
        """
        try:
            data_store = self.health_imaging_client.get_datastore(
                datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't get data store %s. Here's why: %s: %s",
                id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
        else:
            return data_store["datastoreProperties"]
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、GetDatastoreAWS 「 SDK for Python (Boto3) API リファレンス」の「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **GetImageFrame**で を使用する

以下のコード例は、GetImageFrame の使用方法を示しています。

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。

- 画像セットと画像フレームを使い始めます

C++

SDK for C++

```
//! Routine which downloads an AWS HealthImaging image frame.
/*!
  \param dataStoreID: The HealthImaging data store ID.
```

```
  \param imageSetID: The image set ID.
  \param frameID: The image frame ID.
  \param jphFile: File to store the downloaded frame.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageFrame(const Aws::String &dataStoreID,
                                            const Aws::String &imageSetID,
                                            const Aws::String &frameID,
                                            const Aws::String &jphFile,
                                            const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);

    Aws::MedicalImaging::Model::GetImageFrameRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);

    Aws::MedicalImaging::Model::ImageFrameInformation imageFrameInformation;
    imageFrameInformation.SetImageFrameId(frameID);
    request.SetImageFrameInformation(imageFrameInformation);

    Aws::MedicalImaging::Model::GetImageFrameOutcome outcome =
 client.GetImageFrame(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully retrieved image frame." << std::endl;
        auto &buffer = outcome.GetResult().GetImageFrameBlob();

        std::ofstream outfile(jphFile, std::ios::binary);
        outfile << buffer.rdbuf();
    }
    else {
        std::cout << "Error retrieving image frame." <<
 outcome.GetError().GetMessage()
                  << std::endl;

    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「 API リファレンスGetImageFrame」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

画像セットのピクセルデータを取得するには

次の `get-image-frame` コード例では、画像フレームを取得しています。

```
aws medical-imaging get-image-frame \
    --datastore-id "12345678901234567890123456789012" \
    --image-set-id "98765412345612345678907890789012" \
    --image-frame-information imageFrameId=3abf5d5d7ae72f80a0ec81b2c0de3ef4 \
    imageframe.jph
```

注: このコード例には出力は含まれません。これは、 GetImageFrame アクションがピクセルデータのストリームを imageframe.jph ファイルに返すためです。画像フレームのデコードと表示については、「HTJ2K デコードライブラリ」を参照してください。

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットのピクセルデータの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetImageFrame」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
        public static void getMedicalImageSetFrame(MedicalImagingClient
medicalImagingClient,
                        String destinationPath,
                        String datastoreId,
```

```
                        String imagesetId,
                        String imageFrameId) {

                try {
                        GetImageFrameRequest getImageSetMetadataRequest =
GetImageFrameRequest.builder()
                                        .datastoreId(datastoreId)
                                        .imageSetId(imagesetId)

.imageFrameInformation(ImageFrameInformation.builder()

.imageFrameId(imageFrameId)
                                                        .build())
                                        .build();

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

FileSystems.getDefault().getPath(destinationPath));

                        System.out.println("Image frame downloaded to " +
destinationPath);
                } catch (MedicalImagingException e) {
                        System.err.println(e.awsErrorDetails().errorMessage());
                        System.exit(1);
                }
        }
```

- API の詳細については、「 API リファレンスGetImageFrame」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
```

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     contentType: 'application/octet-stream',
  //     imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};
```

- API の詳細については、「 API リファレンスGetImageFrame」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def get_pixel_data(
        self, file_path_to_write, datastore_id, image_set_id, image_frame_id
    ):
        """
        Get an image frame's pixel data.

        :param file_path_to_write: The path to write the image frame's HTJ2K
 encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.health_imaging_client.get_image_frame(
                datastoreId=datastore_id,
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)
        except ClientError as err:
            logger.error(
```

```
                "Couldn't get image frame. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 GetImageFrame AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **GetImageSet**で を使用する

以下のコード例は、GetImageSet の使用方法を示しています。

CLI

AWS CLI

画像セットのプロパティを取得するには

以下の get-image-set コード例では、画像セットのプロパティを取得しています。

```
aws medical-imaging get-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id 18f88ac7870584f58d56256646b4d92b \
```

```
    --version-id 1
```

出力:

```
{
    "versionId": "1",
    "imageSetWorkflowStatus": "COPIED",
    "updatedAt": 1680027253.471,
    "imageSetId": "18f88ac7870584f58d56256646b4d92b",
    "imageSetState": "ACTIVE",
    "createdAt": 1679592510.753,
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットのプロパティの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetImageSet」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static GetImageSetResponse getMedicalImageSet(MedicalImagingClient
medicalImagingClient,
            String datastoreId,
            String imagesetId,
            String versionId) {
        try {
            GetImageSetRequest.Builder getImageSetRequestBuilder =
GetImageSetRequest.builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId);

            if (versionId != null) {
                getImageSetRequestBuilder =
getImageSetRequestBuilder.versionId(versionId);
            }

            return
medicalImagingClient.getImageSet(getImageSetRequestBuilder.build());
```

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

        return null;
    }
```

- API の詳細については、「 API リファレンスGetImageSet」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
```

```
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスGetImageSet」の「」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
```

```
        self.health_imaging_client = health_imaging_client


    def get_image_set(self, datastore_id, image_set_id, version_id=None):
        """
        Get the properties of an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.health_imaging_client.get_image_set(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set = self.health_imaging_client.get_image_set(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
        except ClientError as err:
            logger.error(
                "Couldn't get image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return image_set
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 GetImageSet AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **GetImageSetMetadata**で を使用する

以下のコード例は、GetImageSetMetadata の使用方法を示しています。

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。

- 画像セットと画像フレームを使い始めます

C++

SDK for C++

イメージセットのメタデータを取得するためのユーティリティ関数。

```
//! Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The HealthImaging image set ID.
  \param versionID: The HealthImaging image set version ID, ignored if empty.
  \param outputFilePath: The path where the metadata will be stored as gzipped
 json.
  \param clientConfig: Aws client configuration.
  \\return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
 &outputFilePath,
```

```
                                                    const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    if (!versionID.empty()) {
        request.SetVersionId(versionID);
    }
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
 client.GetImageSetMetadata(
            request);
    if (outcome.IsSuccess()) {
        std::ofstream file(outputFilePath, std::ios::binary);
        auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
        file << metadata.rdbuf();
    }
    else {
        std::cerr << "Failed to get image set metadata: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

イメージセットのメタデータをバージョンなしで取得します。

```
        if (AwsDoc::Medical_Imaging::getImageSetMetadata(dataStoreID, imageSetID,
 "", outputFilePath, clientConfig))
        {
            std::cout << "Successfully retrieved image set metadata." <<
 std::endl;
            std::cout << "Metadata stored in: " << outputFilePath << std::endl;
        }
```

イメージセットのメタデータをバージョン付きで取得します。

```
        if (AwsDoc::Medical_Imaging::getImageSetMetadata(dataStoreID, imageSetID,
 versionID, outputFilePath, clientConfig))
        {
```

```
            std::cout << "Successfully retrieved image set metadata." <<
 std::endl;
            std::cout << "Metadata stored in: " << outputFilePath << std::endl;
        }
```

- API の詳細については、「 API リファレンスGetImageSetMetadata」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

例 1：画像セットのメタデータをバージョンなしで取得するには

次の get-image-set-metadata コード例では、バージョンを指定せずに画像セットのメタデータを取得しています。

注：outfile は必須のパラメータです

```
aws medical-imaging get-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    studymetadata.json.gz
```

返されたメタデータは gzip で圧縮され、studymetadata.json.gz ファイルに保存されます。返された JSON オブジェクトの内容を表示するには、まずオブジェクトを解凍する必要があります。

出力:

```
{
    "contentType": "application/json",
    "contentEncoding": "gzip"
}
```

例 2：画像セットのメタデータをバージョン付きで取得するには

次の get-image-set-metadata コード例では、指定されたバージョンの画像セットのメタデータを取得しています。

注：outfile は必須のパラメータです

```
aws medical-imaging get-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --version-id 1 \
    studymetadata.json.gz
```

返されたメタデータは gzip で圧縮され、studymetadata.json.gz ファイルに保存されます。返された JSON オブジェクトの内容を表示するには、まずオブジェクトを解凍する必要があります。

出力:

```
{
    "contentType": "application/json",
    "contentEncoding": "gzip"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットメタデータの取得」を参照してください。

- API の詳細については、「 コマンドリファレンスGetImageSetMetadata」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
    public static void getMedicalImageSetMetadata(MedicalImagingClient
 medicalImagingClient,
            String destinationPath,
            String datastoreId,
            String imagesetId,
            String versionId) {

        try {
```

```
            GetImageSetMetadataRequest.Builder getImageSetMetadataRequestBuilder
= GetImageSetMetadataRequest.builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId);

            if (versionId != null) {
                getImageSetMetadataRequestBuilder =
getImageSetMetadataRequestBuilder.versionId(versionId);
            }


medicalImagingClient.getImageSetMetadata(getImageSetMetadataRequestBuilder.build(),
                    FileSystems.getDefault().getPath(destinationPath));

            System.out.println("Metadata downloaded to " + destinationPath);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「 API リファレンスGetImageSetMetadata」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

イメージセットのメタデータを取得するためのユーティリティ関数。

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
```

```
 * @param {string} metadataFileName - The name of the file for the gzipped
 metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

イメージセットのメタデータをバージョンなしで取得します。

```
try {
  await getImageSetMetadata(
    "metadata.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012"
  );
} catch (err) {
  console.log("Error", err);
}
```

イメージセットのメタデータをバージョン付きで取得します。

```
try {
  await getImageSetMetadata(
    "metadata2.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.log("Error", err);
}
```

- API の詳細については、「 API リファレンスGetImageSetMetadata」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

イメージセットのメタデータを取得するためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def get_image_set_metadata(
        self, metadata_file, datastore_id, image_set_id, version_id=None
    ):
        """
        Get the metadata of an image set.

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """
        try:
            if version_id:
                image_set_metadata =
 self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:

                image_set_metadata =
 self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
            print(image_set_metadata)
            with open(metadata_file, "wb") as f:
                for chunk in
 image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)

        except ClientError as err:
            logger.error(
                "Couldn't get image metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
```

イメージセットのメタデータをバージョンなしで取得します。

```
                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
```

イメージセットのメタデータをバージョン付きで取得します。

```
                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 GetImageSetMetadata AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **ListDICOMImportJobs**で を使用する

以下のコード例は、ListDICOMImportJobs の使用方法を示しています。

CLI

AWS CLI

DICOM インポートジョブを一覧表示するには

次の list-dicom-import-jobs コード例では、インポートジョブを一覧表示します。

```
aws medical-imaging list-dicom-import-jobs \
    --datastore-id "12345678901234567890123456789012"
```

出力:

```
{
    "jobSummaries": [
        {
            "jobId": "09876543210987654321098765432109",
            "jobName": "my-job",
            "jobStatus": "COMPLETED",
            "datastoreId": "12345678901234567890123456789012",
            "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
            "endedAt": "2022-08-12T11:21:56.504000+00:00",
            "submittedAt": "2022-08-12T11:20:21.734000+00:00"
        }
    ]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「インポートジョブの一覧表示」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ListDICOMImportJobs」を参照してください。

Java

SDK for Java 2.x

```
   public static List<DICOMImportJobSummary>
listDicomImportJobs(MedicalImagingClient medicalImagingClient,
           String datastoreId) {

       try {
           ListDicomImportJobsRequest listDicomImportJobsRequest =
ListDicomImportJobsRequest.builder()
                   .datastoreId(datastoreId)
                   .build();
           ListDicomImportJobsResponse response =
medicalImagingClient.listDICOMImportJobs(listDicomImportJobsRequest);
           return response.jobSummaries();
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return new ArrayList<>();
   }
```

- API の詳細については、「 API リファレンスListDICOMImportJobs」を参照してください。AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
```

```
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/
dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- API の詳細については、「 API リファレンスListDICOMImportJobs」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_dicom_import_jobs(self, datastore_id):
        """
        List the DICOM import jobs.

        :param datastore_id: The ID of the data store.
        :return: The list of jobs.
        """
        try:
            paginator = self.health_imaging_client.get_paginator(
                "list_dicom_import_jobs"
            )
            page_iterator = paginator.paginate(datastoreId=datastore_id)
            job_summaries = []
            for page in page_iterator:
                job_summaries.extend(page["jobSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't list DICOM import jobs. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
        else:
            return job_summaries
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 AWS SDK for Python (Boto3) API リファレンスの「ListDICOMImportJobs」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **ListDatastores**で を使用する

以下のコード例は、ListDatastores の使用方法を示しています。

Bash

AWS CLI Bash スクリプトを使用する

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}
```

```
##############################################################################
# function imaging_list_datastores
#
# List the HealthImaging data stores in the account.
#
# Returns:
#       [[datastore_name, datastore_id, datastore_status]]
#    And:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function imaging_list_datastores() {
  local option OPTARG # Required to use getopts command in a function.
  local error_code
  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_list_datastores"
    echo "Lists the AWS HealthImaging data stores in the account."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "h" option; do
    case "${option}" in
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  local response
  response=$(aws medical-imaging list-datastores \
    --output text \
    --query "datastoreSummaries[*][datastoreName, datastoreId, datastoreStatus]")
  error_code=${?}

  if [[ $error_code -ne 0 ]]; then
```

```
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-datastores operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- API の詳細については、「 コマンドリファレンスListDatastores」の「」を参照してください。 AWS CLI

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

データストアを一覧表示するには

次の `list-datastores` コード例では、利用可能なデータストアを一覧表示しています。

```
aws medical-imaging list-datastores
```

出力:

```
{
    "datastoreSummaries": [
        {
            "datastoreId": "12345678901234567890123456789012",
            "datastoreName": "TestDatastore123",
            "datastoreStatus": "ACTIVE",
            "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
            "createdAt": "2022-11-15T23:33:09.643000+00:00",
```

```
            "updatedAt": "2022-11-15T23:33:09.643000+00:00"
        }
    ]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「データストアの一覧表示」を参照してください。

- API の詳細については、「 コマンドリファレンスListDatastores」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static List<DatastoreSummary>
listMedicalImagingDatastores(MedicalImagingClient medicalImagingClient) {
       try {
           ListDatastoresRequest datastoreRequest =
ListDatastoresRequest.builder()
                   .build();
           ListDatastoresIterable responses =
medicalImagingClient.listDatastoresPaginator(datastoreRequest);
           List<DatastoreSummary> datastoreSummaries = new ArrayList<>();

           responses.stream().forEach(response ->
datastoreSummaries.addAll(response.datastoreSummaries()));

           return datastoreSummaries;
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return null;
   }
```

- API の詳細については、「 API リファレンスListDatastores」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
```

```
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- API の詳細については、「 API リファレンスListDatastores」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_datastores(self):
        """
        List the data stores.

        :return: The list of data stores.
        """
        try:
```

```
            paginator =
self.health_imaging_client.get_paginator("list_datastores")
            page_iterator = paginator.paginate()
            datastore_summaries = []
            for page in page_iterator:
                datastore_summaries.extend(page["datastoreSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't list data stores. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return datastore_summaries
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 ListDatastores AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **`ListImageSetVersions`**で を使用する

以下のコード例は、`ListImageSetVersions` の使用方法を示しています。

CLI

AWS CLI

画像セットバージョンを一覧表示するには

次の list-image-set-versions コード例では、画像セットのバージョン履歴を一覧表示しています。

```
aws medical-imaging list-image-set-versions \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e
```

出力:

```
{
    "imageSetPropertiesList": [
        {
            "ImageSetWorkflowStatus": "UPDATED",
            "versionId": "4",
            "updatedAt": 1680029436.304,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
            "ImageSetWorkflowStatus": "UPDATED",
            "versionId": "3",
            "updatedAt": 1680029163.325,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
            "ImageSetWorkflowStatus": "COPY_FAILED",
            "versionId": "2",
            "updatedAt": 1680027455.944,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "message": "INVALID_REQUEST:  Series of SourceImageSet and
 DestinationImageSet don't match.",
            "createdAt": 1680027126.436
        },
```

```
        {
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "versionId": "1",
            "ImageSetWorkflowStatus": "COPIED",
            "createdAt": 1680027126.436
        }
    ]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットのバージョンを一覧表示する」を参照してください。

- API の詳細については、「 コマンドリファレンスListImageSetVersions」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static List<ImageSetProperties>
listMedicalImageSetVersions(MedicalImagingClient medicalImagingClient,
            String datastoreId,
            String imagesetId) {
        try {
            ListImageSetVersionsRequest getImageSetRequest =
ListImageSetVersionsRequest.builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId)
                    .build();

            ListImageSetVersionsIterable responses = medicalImagingClient
                    .listImageSetVersionsPaginator(getImageSetRequest);
            List<ImageSetProperties> imageSetProperties = new ArrayList<>();
            responses.stream().forEach(response ->
imageSetProperties.addAll(response.imageSetPropertiesList()));

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

```
        return null;
    }
```

- API の詳細については、「 API リファレンスListImageSetVersions」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
```

```
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- API の詳細については、「 API リファレンスListImageSetVersions」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
```

```
        self.health_imaging_client = health_imaging_client


    def list_image_set_versions(self, datastore_id, image_set_id):
        """
        List the image set versions.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :return: The list of image set versions.
        """
        try:
            paginator = self.health_imaging_client.get_paginator(
                "list_image_set_versions"
            )
            page_iterator = paginator.paginate(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
            image_set_properties_list = []
            for page in page_iterator:
                image_set_properties_list.extend(page["imageSetPropertiesList"])
        except ClientError as err:
            logger.error(
                "Couldn't list image set versions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return image_set_properties_list
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 ListImageSetVersions AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **ListTagsForResource**で を使用する

以下のコード例は、ListTagsForResource の使用方法を示しています。

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。

- データストアにタグを付ける
- イメージセットにタグを付ける

CLI

AWS CLI

例 1：データストアリソースのタグを一覧表示するには

次の list-tags-for-resource コード例では、データストアのタグを一覧表示しています。

```
aws medical-imaging list-tags-for-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"
```

出力:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

例 2：画像セットリソースのタグを一覧表示するには

次の list-tags-for-resource コード例では、画像セットのタグを一覧表示しています。

```
aws medical-imaging list-tags-for-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b"
```

出力:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「 によるリソースのタグ付け AWS HealthImaging」を参照してください。

- API の詳細については、「 コマンドリファレンスListTagsForResource」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static ListTagsForResourceResponse
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
           String resourceArn) {
       try {
           ListTagsForResourceRequest listTagsForResourceRequest =
ListTagsForResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .build();

           return
medicalImagingClient.listTagsForResource(listTagsForResourceRequest);
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
```

```
        return null;
    }
```

- API の詳細については、「 API リファレンスListTagsForResource」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
```

```
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスListTagsForResource」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_tags_for_resource(self, resource_arn):
        """
        List the tags for a resource.

        :param resource_arn: The ARN of the resource.
        :return: The list of tags.
        """
        try:
            tags = self.health_imaging_client.list_tags_for_resource(
                resourceArn=resource_arn
            )
        except ClientError as err:
            logger.error(
                "Couldn't list tags for resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return tags["tags"]
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 ListTagsForResource AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **SearchImageSets**で を使用する

以下のコード例は、SearchImageSets の使用方法を示しています。

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。

- 画像セットと画像フレームを使い始めます

C++

SDK for C++

イメージセットを検索するためのユーティリティ関数。

```
//! Routine which searches for image sets based on defined input attributes.
```

```
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param searchCriteria: A search criteria instance.
  \param imageSetResults: Vector to receive the image set IDs.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::searchImageSets(const Aws::String &dataStoreID,
                                              const
 Aws::MedicalImaging::Model::SearchCriteria &searchCriteria,
                                              Aws::Vector<Aws::String>
 &imageSetResults,
                                              const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::SearchImageSetsRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetSearchCriteria(searchCriteria);

    Aws::String nextToken; // Used for paginated results.
    bool result = true;
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::MedicalImaging::Model::SearchImageSetsOutcome outcome =
 client.SearchImageSets(
                request);
        if (outcome.IsSuccess()) {
            for (auto &imageSetMetadataSummary:
 outcome.GetResult().GetImageSetsMetadataSummaries()) {

 imageSetResults.push_back(imageSetMetadataSummary.GetImageSetId());
            }

            nextToken = outcome.GetResult().GetNextToken();
        }
        else {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            result = false;
        }
    } while (!nextToken.empty());
```

```
    return result;
}
```

ユースケース #1: EQUAL 演算子。

```
       Aws::Vector<Aws::String> imageIDsForPatientID;
       Aws::MedicalImaging::Model::SearchCriteria searchCriteriaEqualsPatientID;
       Aws::Vector<Aws::MedicalImaging::Model::SearchFilter>
patientIDSearchFilters = {

Aws::MedicalImaging::Model::SearchFilter().WithOperator(Aws::MedicalImaging::Model::Oper

.WithValues({Aws::MedicalImaging::Model::SearchByAttributeValue().WithDICOMPatientId(pat
       };

       searchCriteriaEqualsPatientID.SetFilters(patientIDSearchFilters);
       bool result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,

searchCriteriaEqualsPatientID,

imageIDsForPatientID,
                                                                clientConfig);
       if (result) {
           std::cout << imageIDsForPatientID.size() << " image sets found for
the patient with ID '"
           <<  patientID << "'." << std::endl;
           for (auto &imageSetResult : imageIDsForPatientID) {
               std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
           }
       }
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
        Aws::MedicalImaging::Model::SearchByAttributeValue useCase2StartDate;

useCase2StartDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAnd
       .WithDICOMStudyDate("19990101")
       .WithDICOMStudyTime("000000.000"));
```

```
        Aws::MedicalImaging::Model::SearchByAttributeValue useCase2EndDate;

 useCase2EndDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAndTi

  .WithDICOMStudyDate(Aws::Utils::DateTime(std::chrono::system_clock::now()).ToLocalTimeSt
%m%d"))
        .WithDICOMStudyTime("000000.000"));

        Aws::MedicalImaging::Model::SearchFilter useCase2SearchFilter;
        useCase2SearchFilter.SetValues({useCase2StartDate, useCase2EndDate});

 useCase2SearchFilter.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchCriteria useCase2SearchCriteria;
        useCase2SearchCriteria.SetFilters({useCase2SearchFilter});

        Aws::Vector<Aws::String> usesCase2Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase2SearchCriteria,
                                                          usesCase2Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase2Results.size() << " image sets found for
 between 1999/01/01 and present."
                      <<  std::endl;
            for (auto &imageSetResult : usesCase2Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
 std::endl;
            }
        }
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
        Aws::MedicalImaging::Model::SearchByAttributeValue useCase3StartDate;

 useCase3StartDate.SetCreatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase3EndDate;

 useCase3EndDate.SetCreatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));
```

```
        Aws::MedicalImaging::Model::SearchFilter useCase3SearchFilter;
        useCase3SearchFilter.SetValues({useCase3StartDate, useCase3EndDate});

useCase3SearchFilter.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchCriteria useCase3SearchCriteria;
        useCase3SearchCriteria.SetFilters({useCase3SearchFilter});

        Aws::Vector<Aws::String> usesCase3Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase3SearchCriteria,
                                                          usesCase3Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase3Results.size() << " image sets found for
created between 2023/11/30 and present."
                      <<  std::endl;
            for (auto &imageSetResult : usesCase3Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順序でのソートレスポンス。

```
        Aws::MedicalImaging::Model::SearchByAttributeValue useCase4StartDate;

useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase4EndDate;

useCase4EndDate.SetUpdatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterBetween;
        useCase4SearchFilterBetween.SetValues({useCase4StartDate,
useCase4EndDate});

useCase4SearchFilterBetween.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchByAttributeValue seriesInstanceUID;
```

```
        seriesInstanceUID.SetDICOMSeriesInstanceUID(dicomSeriesInstanceUID);

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterEqual;
        useCase4SearchFilterEqual.SetValues({seriesInstanceUID});

useCase4SearchFilterEqual.SetOperator(Aws::MedicalImaging::Model::Operator::EQUAL);

        Aws::MedicalImaging::Model::SearchCriteria useCase4SearchCriteria;
        useCase4SearchCriteria.SetFilters({useCase4SearchFilterBetween,
useCase4SearchFilterEqual});

        Aws::MedicalImaging::Model::Sort useCase4Sort;

useCase4Sort.SetSortField(Aws::MedicalImaging::Model::SortField::updatedAt);
        useCase4Sort.SetSortOrder(Aws::MedicalImaging::Model::SortOrder::ASC);

        useCase4SearchCriteria.SetSort(useCase4Sort);

        Aws::Vector<Aws::String> usesCase4Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase4SearchCriteria,
                                                          usesCase4Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase4Results.size() << " image sets found for EQUAL
operator "
            << "on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort
response\n"
            <<  "in ASC order on updatedAt field." <<  std::endl;
            for (auto &imageSetResult : usesCase4Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

- API の詳細については、「 API リファレンスSearchImageSets」の「」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

例 1 : EQUAL 演算子を使用して画像セットを検索するには

次の search-image-sets コード例では、EQUAL 演算子を使用し、特定の値に基づいて画像セットを検索しています。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria file://search-criteria.json
```

search-criteria.json の内容

```
{
    "filters": [{
        "values": [{"DICOMPatientId" : "SUBJECT08701"}],
        "operator": "EQUAL"
    }]
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
             "DICOMPatientSex": "F",
             "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
             "DICOMPatientBirthDate": "19201120",
```

```
                "DICOMStudyDescription": "UNKNOWN",
                "DICOMPatientId": "SUBJECT08701",
                "DICOMPatientName": "Melissa844 Huel628",
                "DICOMNumberOfStudyRelatedInstances": 1,
                "DICOMStudyTime": "140728",
                "DICOMNumberOfStudyRelatedSeries": 1
            },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

例 2: DICOM StudyDate と DICOM を使用して BETWEEN 演算子で画像セットを検索するにはStudyTime

次の `search-image-sets` コード例では、1990 年 1 月 1 日 (午前 0 時) から 2023 年 1 月 1 日 (午前 0 時) の間に生成された DICOM スタディを含む画像セットを検索します。

注: DICOM StudyTime はオプションです。入力されていない場合は、フィルターで指定された日付の時間値は午前 0 時 (1 日の始まり) になります。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria file://search-criteria.json
```

`search-criteria.json` の内容

```
{
    "filters": [{
        "values": [{
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "19900101",
                "DICOMStudyTime": "000000"
            }
        },
        {
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "20230101",
                "DICOMStudyTime": "000000"
            }
        }],
        "operator": "BETWEEN"
    }]
```

```
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

例 3：createdAt を使用して BETWEEN 演算子を使用して画像セットを検索するには (スタディが以前に保存されていた時間)

次のsearch-image-setsコード例では、UTC タイムゾーンの時間範囲内に DICOM スタディ HealthImaging が保持されている画像セットを検索します。

注：createdAt をサンプル形式 ("1985-04-12T23:20:50.52Z") で提供してください。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria  file://search-criteria.json
```

search-criteria.json の内容

```
{
    "filters": [{
```

```
        "values": [{
            "createdAt": "1985-04-12T23:20:50.52Z"
        },
        {
            "createdAt": "2022-04-12T23:20:50.52Z"
        }],
        "operator": "BETWEEN"
    }]
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

例 4: DICOM SeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN を使用して画像セットを検索し、updatedAt フィールドで ASC 順序でレスポンスをソートするには

次のsearch-image-setsコード例では、DICOM SeriesInstanceUID に EQUAL 演算子を使用し、updatedAt に BETWEEN を指定し、updatedAt フィールドに ASC 順でソートレスポンスを指定して画像セットを検索します。

注: updatedAt をサンプル形式 ("1985-04-12T23:20:50.52Z") で指定します。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria  file://search-criteria.json
```

search-criteria.json の内容

```
{
    "filters": [{
        "values": [{
            "updatedAt": "2024-03-11T15:00:05.074000-07:00"
        }, {
            "updatedAt": "2024-03-11T16:00:05.074000-07:00"
        }],
        "operator": "BETWEEN"
    }, {
        "values": [{
            "DICOMSeriesInstanceUID": "1.2.840.99999999.84710745.943275268089"
        }],
        "operator": "EQUAL"
    }],
    "sort": {
        "sortField": "updatedAt",
        "sortOrder": "ASC"
    }
}
```

出力:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
```

```
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「イメージセットの検索」を参照してください。

- API の詳細については、「 コマンドリファレンスSearchImageSets」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

イメージセットを検索するためのユーティリティ関数。

```
   public static List<ImageSetsMetadataSummary> searchMedicalImagingImageSets(
           MedicalImagingClient medicalImagingClient,
           String datastoreId, SearchCriteria searchCriteria) {
       try {
           SearchImageSetsRequest datastoreRequest =
SearchImageSetsRequest.builder()
                   .datastoreId(datastoreId)
                   .searchCriteria(searchCriteria)
                   .build();
           SearchImageSetsIterable responses = medicalImagingClient
                   .searchImageSetsPaginator(datastoreRequest);
           List<ImageSetsMetadataSummary> imageSetsMetadataSummaries = new
ArrayList<>();

           responses.stream().forEach(response -> imageSetsMetadataSummaries
                   .addAll(response.imageSetsMetadataSummaries()));

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

```
        return null;
    }
```

ユースケース #1: EQUAL 演算子。

```
        List<SearchFilter> searchFilters =
 Collections.singletonList(SearchFilter.builder()
                .operator(Operator.EQUAL)
                .values(SearchByAttributeValue.builder()
                        .dicomPatientId(patientId)
                        .build())
                .build());

        SearchCriteria searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .build();

        List<ImageSetsMetadataSummary> imageSetsMetadataSummaries =
 searchMedicalImagingImageSets(
                medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println("The image sets for patient " + patientId + " are:
\n"
                    + imageSetsMetadataSummaries);
            System.out.println();
        }
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMdd");
        searchFilters = Collections.singletonList(SearchFilter.builder()
                .operator(Operator.BETWEEN)
                .values(SearchByAttributeValue.builder()

 .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                                        .dicomStudyDate("19990101")
                                        .dicomStudyTime("000000.000")
                                        .build())
                                .build(),
                        SearchByAttributeValue.builder()
```

```
.dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                                        .dicomStudyDate((LocalDate.now()
                                                .format(formatter)))
                                        .dicomStudyTime("000000.000")
                                        .build())
                                .build())
                .build());

        searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .build();

        imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println(
                    "The image sets searched with BETWEEN operator using
DICOMStudyDate and DICOMStudyTime are:\n"
                            +
                            imageSetsMetadataSummaries);
            System.out.println();
        }
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
        searchFilters = Collections.singletonList(SearchFilter.builder()
                .operator(Operator.BETWEEN)
                .values(SearchByAttributeValue.builder()

.createdAt(Instant.parse("1985-04-12T23:20:50.52Z"))
                                .build(),
                        SearchByAttributeValue.builder()
                                .createdAt(Instant.now())
                                .build())
                .build());

        searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .build();
```

```
        imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println("The image sets searched with BETWEEN operator
using createdAt are:\n "
                    + imageSetsMetadataSummaries);
            System.out.println();
        }
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順序でのソートレスポンス。

```
        Instant startDate = Instant.parse("1985-04-12T23:20:50.52Z");
        Instant endDate = Instant.now();

        searchFilters = Arrays.asList(
                SearchFilter.builder()
                        .operator(Operator.EQUAL)
                        .values(SearchByAttributeValue.builder()
                                .dicomSeriesInstanceUID(seriesInstanceUID)
                                .build())
                        .build(),
                SearchFilter.builder()
                        .operator(Operator.BETWEEN)
                        .values(

SearchByAttributeValue.builder().updatedAt(startDate).build(),

SearchByAttributeValue.builder().updatedAt(endDate).build()
                        ).build());

        Sort sort =
Sort.builder().sortOrder(SortOrder.ASC).sortField(SortField.UPDATED_AT).build();

        searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .sort(sort)
                .build();

        imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
```

```
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println("The image sets searched with EQUAL operator on
 DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response\n" +
                    "in ASC order on updatedAt field are:\n "
                    + imageSetsMetadataSummaries);
            System.out.println();
        }
```

- API の詳細については、「 API リファレンスSearchImageSets」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

イメージセットを検索するためのユーティリティ関数。

```
import {paginateSearchImageSets} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters -
 The search criteria filters.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
 criteria sort.
 */
export const searchImageSets = async (
    datastoreId = "xxxxxxxx",
    searchCriteria = {}
) => {
    const paginatorConfig = {
        client: medicalImagingClient,
        pageSize: 50,
```

```
    };

    const commandParams = {
        datastoreId: datastoreId,
        searchCriteria: searchCriteria,
    };

    const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

    const imageSetsMetadataSummaries = [];
    for await (const page of paginator) {
        // Each page contains a list of `jobSummaries`. The list is truncated if
 is larger than `pageSize`.
        imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
        console.log(page);
    }
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //     imageSetsMetadataSummaries: [
    //         {
    //             DICOMTags: [Object],
    //             createdAt: "2023-09-19T16:59:40.551Z",
    //             imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
    //             updatedAt: "2023-09-19T16:59:40.551Z",
    //             version: 1
    //         }]
    // }

    return imageSetsMetadataSummaries;
};
```

ユースケース #1: EQUAL 演算子。

```
    const datastoreId = "12345678901234567890123456789012";
```

```
    try {
        const searchCriteria = {
            filters: [
                {
                    values: [{DICOMPatientId: "1234567"}],
                    operator: "EQUAL",
                },
            ]
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {
                            DICOMStudyDateAndTime: {
                                DICOMStudyDate: "19900101",
                                DICOMStudyTime: "000000",
                            },
                        },
                        {
                            DICOMStudyDateAndTime: {
                                DICOMStudyDate: "20230901",
                                DICOMStudyTime: "000000",
                            },
                        },
                    ],
                    operator: "BETWEEN",
                },
            ]
        };
```

```
    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
const datastoreId = "1234567890123456789012345678901 2";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {createdAt: new Date("1985-04-12T23:20:50.52Z")},
                    {createdAt: new Date()},
                ],
                operator: "BETWEEN",
            },
        ]
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順序でのソートレスポンス。

```
const datastoreId = "1234567890123456789012345678901 2";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
                    {updatedAt: new Date()},
```

```
                    ],
                    operator: "BETWEEN",
                },
                {
                    values: [
                        {DICOMSeriesInstanceUID:
 "1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
                    ],
                    operator: "EQUAL",
                },
            ],
            sort: {
                sortOrder: "ASC",
                sortField: "updatedAt",
            }
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

- API の詳細については、「 API リファレンスSearchImageSets」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

イメージセットを検索するためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client
```

```
   def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.health_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries
```

ユースケース #1: EQUAL 演算子。

```
        search_filter = {
            "filters": [
                {"operator": "EQUAL", "values": [{"DICOMPatientId": patient_id}]}
            ]
        }

        image_sets = self.search_image_sets(data_store_id, search_filter)
        print(f"Image sets found with EQUAL operator\n{image_sets}")
```

ユースケース #2: DICOM StudyDate と DICOM を使用する BETWEEN 演算子StudyTime。

```
        search_filter = {
            "filters": [
                {
                    "operator": "BETWEEN",
                    "values": [
                        {
                            "DICOMStudyDateAndTime": {
                                "DICOMStudyDate": "19900101",
                                "DICOMStudyTime": "000000",
                            }
                        },
                        {
                            "DICOMStudyDateAndTime": {
                                "DICOMStudyDate": "20230101",
                                "DICOMStudyTime": "000000",
                            }
                        },
                    ],
                }
            ]
        }

        image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            f"Image sets found with BETWEEN operator using DICOMStudyDate and
DICOMStudyTime\n{image_sets}"
        )
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
        search_filter = {
            "filters": [
                {
                    "values": [
                        {
                            "createdAt": datetime.datetime(
                                2021, 8, 4, 14, 49, 54, 429000
                            )
                        },
```

```
                        {
                            "createdAt": datetime.datetime.now()
                            + datetime.timedelta(days=1)
                        },
                    ],
                    "operator": "BETWEEN",
                }
            ]
        }

        recent_image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            f"Image sets found with with BETWEEN operator using createdAt
\n{recent_image_sets}"
        )
```

ユースケース #4: DICOM SeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN、および updatedAt フィールドの ASC 順序でのソートレスポンス。

```
        search_filter = {
            "filters": [
                {
                    "values": [
                        {
                            "updatedAt": datetime.datetime(
                                2021, 8, 4, 14, 49, 54, 429000
                            )
                        },
                        {
                            "updatedAt": datetime.datetime.now()
                            + datetime.timedelta(days=1)
                        },
                    ],
                    "operator": "BETWEEN",
                },
                {
                    "values": [{"DICOMSeriesInstanceUID": series_instance_uid}],
                    "operator": "EQUAL",
                },
            ],
            "sort": {
                "sortOrder": "ASC",
```

```
                "sortField": "updatedAt",
            },
        }

        image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            "Image sets found with EQUAL operator on DICOMSeriesInstanceUID and
BETWEEN on updatedAt and"
        )
        print(f"sort response in ASC order on updatedAt field\n{image_sets}")
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 SearchImageSets AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **StartDICOMImportJob**で を使用する

以下のコード例は、StartDICOMImportJob の使用方法を示しています。

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。

- 画像セットと画像フレームを使い始めます

C++

SDK for C++

```
//! Routine which starts a HealthImaging import job.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param inputBucketName: The name of the Amazon S3 bucket containing the DICOM
 files.
  \param inputDirectory: The directory in the S3 bucket containing the DICOM
 files.
  \param outputBucketName: The name of the S3 bucket for the output.
  \param outputDirectory: The directory in the S3 bucket to store the output.
  \param roleArn: The ARN of the IAM role with permissions for the import.
  \param importJobId: A string to receive the import job ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::startDICOMImportJob(
        const Aws::String &dataStoreID, const Aws::String &inputBucketName,
        const Aws::String &inputDirectory, const Aws::String &outputBucketName,
        const Aws::String &outputDirectory, const Aws::String &roleArn,
        Aws::String &importJobId,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(clientConfig);
    Aws::String inputURI = "s3://" + inputBucketName + "/" + inputDirectory +
 "/";
    Aws::String outputURI = "s3://" + outputBucketName + "/" + outputDirectory +
 "/";
    Aws::MedicalImaging::Model::StartDICOMImportJobRequest
 startDICOMImportJobRequest;
    startDICOMImportJobRequest.SetDatastoreId(dataStoreID);
    startDICOMImportJobRequest.SetDataAccessRoleArn(roleArn);
    startDICOMImportJobRequest.SetInputS3Uri(inputURI);
    startDICOMImportJobRequest.SetOutputS3Uri(outputURI);

    Aws::MedicalImaging::Model::StartDICOMImportJobOutcome
 startDICOMImportJobOutcome = medicalImagingClient.StartDICOMImportJob(
            startDICOMImportJobRequest);

    if (startDICOMImportJobOutcome.IsSuccess()) {
        importJobId = startDICOMImportJobOutcome.GetResult().GetJobId();
    }
    else {
```

```
        std::cerr << "Failed to start DICOM import job because "
                  << startDICOMImportJobOutcome.GetError().GetMessage() <<
 std::endl;
    }

    return startDICOMImportJobOutcome.IsSuccess();
}
```

- API の詳細については、「 API リファレンス」のStartDICOMImportJob」を参照してください。 AWS SDK for C++

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CLI

AWS CLI

DICOM インポートジョブを開始するには

次の `start-dicom-import-job` コード例では、DICOM インポートジョブを開始しています。

```
aws medical-imaging start-dicom-import-job \
    --job-name "my-job" \
    --datastore-id "12345678901234567890123456789012" \
    --input-s3-uri "s3://medical-imaging-dicom-input/dicom_input/" \
    --output-s3-uri "s3://medical-imaging-output/job_output/" \
    --data-access-role-arn "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole"
```

出力:

```
{
    "datastoreId": "12345678901234567890123456789012",
    "jobId": "09876543210987654321098765432109",
```

```
    "jobStatus": "SUBMITTED",
    "submittedAt": "2022-08-12T11:28:11.152000+00:00"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「インポートジョブの開始」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StartDICOMImportJob」を参照してください。

Java

SDK for Java 2.x

```
   public static String startDicomImportJob(MedicalImagingClient
medicalImagingClient,
           String jobName,
           String datastoreId,
           String dataAccessRoleArn,
           String inputS3Uri,
           String outputS3Uri) {

       try {
           StartDicomImportJobRequest startDicomImportJobRequest =
StartDicomImportJobRequest.builder()
                   .jobName(jobName)
                   .datastoreId(datastoreId)
                   .dataAccessRoleArn(dataAccessRoleArn)
                   .inputS3Uri(inputS3Uri)
                   .outputS3Uri(outputS3Uri)
                   .build();
           StartDicomImportJobResponse response =
medicalImagingClient.startDICOMImportJob(startDicomImportJobRequest);
           return response.jobId();
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return "";
   }
```

- API の詳細については、「 API リファレンス」のStartDICOMImportJob」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input
 files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files
 are stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
```

```
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- API の詳細については、「 API リファレンス」のStartDICOMImportJob」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def start_dicom_import_job(
        self, job_name, datastore_id, role_arn, input_s3_uri, output_s3_uri
    ):
        """
        Start a DICOM import job.
```

```
        :param job_name: The name of the job.
        :param datastore_id: The ID of the data store.
        :param role_arn: The Amazon Resource Name (ARN) of the role to use for
the job.
        :param input_s3_uri: The S3 bucket input prefix path containing the DICOM
files.
        :param output_s3_uri: The S3 bucket output prefix path for the result.
        :return: The job ID.
        """
        try:
            job = self.health_imaging_client.start_dicom_import_job(
                jobName=job_name,
                datastoreId=datastore_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_s3_uri,
                outputS3Uri=output_s3_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 AWS SDK for StartDICOMImportJob」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **TagResource**で を使用する

以下のコード例は、TagResource の使用方法を示しています。

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。

- データストアにタグを付ける
- イメージセットにタグを付ける

CLI

AWS CLI

例 1：データストアにタグを付けるには

次の tag-resource コード例では、データストアにタグを付けています。

```
aws medical-imaging tag-resource \
  --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
  --tags '{"Deployment":"Development"}'
```

このコマンドでは何も出力されません。

例 2：画像セットにタグを付けるには

次の tag-resource コード例では、画像セットにタグを付けています。

```
aws medical-imaging tag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
    --tags '{"Deployment":"Development"}'
```

このコマンドでは何も出力されません。

詳細については、「 AWS HealthImaging デベロッパーガイド」の「 によるリソースのタグ付け AWS HealthImaging」を参照してください。

- API の詳細については、「 コマンドリファレンスTagResource」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void tagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Map<String, String> tags) {
       try {
           TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tags(tags)
                   .build();

           medicalImagingClient.tagResource(tagResourceRequest);

           System.out.println("Tags have been added to the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「 API リファレンスTagResource」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスTagResource」の「」を参照してください。AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def tag_resource(self, resource_arn, tags):
        """
        Tag a resource.

        :param resource_arn: The ARN of the resource.
        :param tags: The tags to apply.
        """
        try:
            self.health_imaging_client.tag_resource(resourceArn=resource_arn,
 tags=tags)
        except ClientError as err:
            logger.error(
                "Couldn't tag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 TagResource AWS SDK for Python (Boto3) API リファレンス の「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 `UntagResource`で を使用する

以下のコード例は、UntagResource の使用方法を示しています。

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。

- データストアにタグを付ける
- イメージセットにタグを付ける

CLI

AWS CLI

例 1：データストアのタグを削除するには

次の untag-resource コード例では、データストアにタグを削除します。

```
aws medical-imaging untag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
    --tag-keys '["Deployment"]'
```

このコマンドでは何も出力されません。

例 2：画像セットにタグを削除するには

次の untag-resource コード例では、画像セットにタグを削除します。

```
aws medical-imaging untag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
    --tag-keys '["Deployment"]'
```

このコマンドでは何も出力されません。

詳細については、「 AWS HealthImaging デベロッパーガイド」の「 によるリソースのタグ付け AWS HealthImaging」を参照してください。

- API の詳細については、「 コマンドリファレンスUntagResource」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void untagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Collection<String> tagKeys) {
       try {
           UntagResourceRequest untagResourceRequest =
UntagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tagKeys(tagKeys)
                   .build();

           medicalImagingClient.untagResource(untagResourceRequest);

           System.out.println("Tags have been removed from the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「 API リファレンスUntagResource」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API の詳細については、「 API リファレンスUntagResource」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def untag_resource(self, resource_arn, tag_keys):
        """
        Untag a resource.

        :param resource_arn: The ARN of the resource.
        :param tag_keys: The tag keys to remove.
        """
        try:
            self.health_imaging_client.untag_resource(
                resourceArn=resource_arn, tagKeys=tag_keys
            )
        except ClientError as err:
            logger.error(
                "Couldn't untag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
```

```
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API の詳細については、 UntagResource AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 **UpdateImageSetMetadata**で を使用する

以下のコード例は、UpdateImageSetMetadata の使用方法を示しています。

CLI

AWS CLI

画像セットメタデータに属性を挿入または更新するには

次のupdate-image-set-metadataコード例では、画像セットメタデータに属性を挿入または更新します。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
    "DICOMUpdates": {
        "updatableAttributes":
 "eyJTY2hlbWFWZXJzaW9uIjoxLjEsIlBhdGllbnQiOnsiRElDT00iOnsiUGF0aWVudE5hbWUiOiJNWF5NWCJ9fX0
```

```
    }
}
```

注：updatableAttributes は Base64 でエンコードされた JSON 文字列です。こちらはエンコードされていない JSON 文字列です。

{"SchemaVersion":1.1,"""":{"DICOM":{"PatientName""MX^MX"}}}

出力:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

画像セットメタデータから属性を削除するには

次のupdate-image-set-metadataコード例では、画像セットメタデータから属性を削除します。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
    "DICOMUpdates": {
        "removableAttributes":
 "e1NjaGVtYVZlcnNpb246MS4xLFN0dWR5OntESUNPTTp7U3R1ZHlEZXNjcmlwdGlvbjpDSEVTVH19fQo="
    }
}
```

注：removableAttributes は Base64 でエンコードされた JSON 文字列です。こちらはエンコードされていない JSON 文字列です。キーと値は、削除する属性と一致する必要があります。

{SchemaVersion"":1.1,"Study":{"DICOM":{"StudyDescription""CHEST"}}}

出力:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

イメージセットメタデータからインスタンスを削除するには

次のupdate-image-set-metadataコード例では、イメージセットメタデータからインスタンスを削除します。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
    "DICOMUpdates": {
        "removableAttributes":
 "eezEuMS4xLjEuMS4xLjEyMzQ1LjEyMzQ1Njc4OTAxMi4xMjMuMTIzNDU2Nzg5MDEyMzQuMTp7SW5zdGFuY2VzOn
    }
}
```

注：removableAttributes は Base64 でエンコードされた JSON 文字列です。こちらはエンコードされていない JSON 文字列です。

{"1.1.1.1.1.12345.123456789012.123.12345678901234.1":{"Instances":
{"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1":{}}}}}

出力:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「 AWS HealthImaging デベロッパーガイド」の「画像セットメタデータの更新」を参照してください。

- API の詳細については、「 コマンドリファレンスUpdateImageSetMetadata」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

```
   public static void updateMedicalImageSetMetadata(MedicalImagingClient
medicalImagingClient,
                                                     String datastoreId,
                                                     String imagesetId,
                                                     String versionId,
                                                     MetadataUpdates
metadataUpdates) {
       try {
           UpdateImageSetMetadataRequest updateImageSetMetadataRequest =
UpdateImageSetMetadataRequest
                   .builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId)
                   .latestVersionId(versionId)
                   .updateImageSetMetadataUpdates(metadataUpdates)
                   .build();
```

```
            UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

            System.out.println("The image set metadata was updated" + response);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

ユースケース #1: 属性を挿入または更新します。

```
            final String insertAttributes = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "DICOM": {
                          "StudyDescription": "CT CHEST"
                        }
                      }
                    }
                    """;
            MetadataUpdates metadataInsertUpdates = MetadataUpdates.builder()
                    .dicomUpdates(DICOMUpdates.builder()
                            .updatableAttributes(SdkBytes.fromByteBuffer(
                                    ByteBuffer.wrap(insertAttributes
                                            .getBytes(StandardCharsets.UTF_8))))
                            .build())
                    .build();

            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                    versionid, metadataInsertUpdates);
```

ユースケース #2: 属性を削除します。

```
            final String removeAttributes = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "DICOM": {
```

```
                        "StudyDescription": "CT CHEST"
                      }
                    }
                  }
                  """;
          MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                  .dicomUpdates(DICOMUpdates.builder()
                          .removableAttributes(SdkBytes.fromByteBuffer(
                                  ByteBuffer.wrap(removeAttributes
                                          .getBytes(StandardCharsets.UTF_8))))
                          .build())
                  .build();

          updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                  versionid, metadataRemoveUpdates);
```

ユースケース #3: インスタンスを削除します。

```
          final String removeInstance = """
                  {
                    "SchemaVersion": 1.1,
                    "Study": {
                      "Series": {
                        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1":
{
                          "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                          }
                        }
                      }
                    }
                  }
                  """;
          MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                  .dicomUpdates(DICOMUpdates.builder()
                          .removableAttributes(SdkBytes.fromByteBuffer(
                                  ByteBuffer.wrap(removeInstance
                                          .getBytes(StandardCharsets.UTF_8))))
                          .build())
                  .build();
```

```
            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
 imagesetId,
                    versionid, metadataRemoveUpdates);
```

- API の詳細については、「 API リファレンスUpdateImageSetMetadata」の「」を参照してください。 AWS SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

```
import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    console.log(response);
```

```
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
};
```

ユースケース #1: 属性を挿入または更新します。

```
        const insertAttributes =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "updatableAttributes":
                    new TextEncoder().encode(insertAttributes)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

ユースケース #2: 属性を削除します。

```
        // Attribute key and value must match the existing attribute.
        const remove_attribute =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "removableAttributes":
                    new TextEncoder().encode(remove_attribute)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

ユースケース #3: インスタンスを削除します。

```
        const remove_instance =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "Series": {
                        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
                        }
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
```

```
                "removableAttributes":
                    new TextEncoder().encode(remove_instance)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

- API の詳細については、「 API リファレンスUpdateImageSetMetadata」の「」を参照してください。 AWS SDK for JavaScript

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def update_image_set_metadata(
        self, datastore_id, image_set_id, version_id, metadata
    ):
        """
        Update the metadata of an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param metadata: The image set metadata as a dictionary.
            For example {"DICOMUpdates": {"updatableAttributes":
            "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{\"PatientName\":
\"Garcia^Gloria\"}}}"}}
        :return: The updated image set metadata.
        """
```

```
        try:
            updated_metadata =
self.health_imaging_client.update_image_set_metadata(
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                latestVersionId=version_id,
                updateImageSetMetadataUpdates=metadata,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update image set metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return updated_metadata
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

ユースケース #1: 属性を挿入または更新します。

```
            attributes = """{
                    "SchemaVersion": 1.1,
                    "Study": {
                        "DICOM": {
                            "StudyDescription": "CT CHEST"
                        }
                    }
                }"""
            metadata = {"DICOMUpdates": {"updatableAttributes": attributes}}

            self.update_image_set_metadata(
                data_store_id, image_set_id, version_id, metadata
            )
```

ユースケース #2: 属性を削除します。

```
            # Attribute key and value must match the existing attribute.
            attributes = """{
                    "SchemaVersion": 1.1,
                    "Study": {
                        "DICOM": {
                            "StudyDescription": "CT CHEST"
                        }
                    }
                }"""
            metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

            self.update_image_set_metadata(
                data_store_id, image_set_id, version_id, metadata
            )
```

ユースケース #3: インスタンスを削除します。

```
            attributes = """{
                    "SchemaVersion": 1.1,
                    "Study": {
                        "Series": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                                "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                                }
                            }
                        }
                    }
                }"""
            metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

            self.update_image_set_metadata(
                data_store_id, image_set_id, version_id, metadata
            )
```

- API の詳細については、 UpdateImageSetMetadata AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 HealthImaging を使用するシナリオ

次のコード例は、 AWS SDKs を使用して で HealthImaging一般的なシナリオを実装する方法を示しています。これらのシナリオは、 内で複数の関数を呼び出して特定のタスクを実行する方法を示しています HealthImaging。各シナリオには GitHub、コードの設定と実行の手順を示す へのリンクが含まれています。

例

- AWS SDK を使用して HealthImaging 画像セットと画像フレームの使用を開始する
- AWS SDK を使用した HealthImaging データストアのタグ付け
- AWS SDK を使用した HealthImaging 画像セットのタグ付け

AWS SDK を使用して HealthImaging 画像セットと画像フレームの使用を開始する

次のコード例は、DICOM ファイルをインポートし、画像フレームを にダウンロードする方法を示しています HealthImaging。

実装は、ワークフローのコマンドラインアプリケーションとして構造化されています。

- DICOM インポートするためのリソースのセットアップ
- DICOM ファイルをデータストアへのインポート。
- インポートジョブの画像セット ID の取得。
- インポートジョブの画像フレーム ID の取得。

- イメージフレームをダウンロード、デコード、および検証します。
- リソースをクリーンアップします。

C++

SDK for C++

必要なリソースを持つ AWS CloudFormation スタックを作成します。

```
   Aws::String inputBucketName;
   Aws::String outputBucketName;
   Aws::String dataStoreId;
   Aws::String roleArn;
   Aws::String stackName;

   if (askYesNoQuestion(
           "Would you like to let this workflow create the resources for you?
(y/n) ")) {
       stackName = askQuestion(
               "Enter a name for the AWS CloudFormation stack to create. ");
       Aws::String dataStoreName = askQuestion(
               "Enter a name for the HealthImaging datastore to create. ");

       Aws::Map<Aws::String, Aws::String> outputs = createCloudFormationStack(
               stackName,
               dataStoreName,
               clientConfiguration);

       if (!retrieveOutputs(outputs, dataStoreId, inputBucketName,
outputBucketName,
                            roleArn)) {
           return false;
       }

       std::cout << "The following resources have been created." << std::endl;
       std::cout << "A HealthImaging datastore with ID: " << dataStoreId << "."
                 << std::endl;
       std::cout << "An Amazon S3 input bucket named: " << inputBucketName <<
"."
                 << std::endl;
       std::cout << "An Amazon S3 output bucket named: " << outputBucketName <<
"."
```

```
                  << std::endl;
        std::cout << "An IAM role with the ARN: " << roleArn << "." << std::endl;
        askQuestion("Enter return to continue.", alwaysTrueTest);
    }
    else {
        std::cout << "You have chosen to use preexisting resources:" <<
std::endl;
        dataStoreId = askQuestion(
                "Enter the data store ID of the HealthImaging datastore you wish
to use: ");
        inputBucketName = askQuestion(
                "Enter the name of the S3 input bucket you wish to use: ");
        outputBucketName = askQuestion(
                "Enter the name of the S3 output bucket you wish to use: ");
        roleArn = askQuestion(
                "Enter the ARN for the IAM role with the proper permissions to
import a DICOM series: ");
    }
```

DICOM ファイルを Amazon S3 インポートバケットにコピーします。

```
    std::cout
            << "This workflow uses DICOM files from the National Cancer Institute
Imaging Data\n"
            << "Commons (IDC) Collections." << std::endl;
    std::cout << "Here is the link to their website." << std::endl;
    std::cout << "https://registry.opendata.aws/nci-imaging-data-commons/" <<
std::endl;
    std::cout << "We will use DICOM files stored in an S3 bucket managed by the
IDC."
              << std::endl;
    std::cout
            << "First one of the DICOM folders in the IDC collection must be
copied to your\n"
               "input S3 bucket."
            << std::endl;
    std::cout << "You have the choice of one of the following "
              << IDC_ImageChoices.size() << " folders to copy." << std::endl;

    int index = 1;
    for (auto &idcChoice: IDC_ImageChoices) {
        std::cout << index << " - " << idcChoice.mDescription << std::endl;
```

```
        index++;
    }
    int choice = askQuestionForIntRange("Choose DICOM files to import: ", 1, 4);

    Aws::String fromDirectory = IDC_ImageChoices[choice - 1].mDirectory;
    Aws::String inputDirectory = "input";

    std::cout << "The files in the directory '" << fromDirectory << "' in the
 bucket '"
              << IDC_S3_BucketName << "' will be copied " << std::endl;
    std::cout << "to the folder '" << inputDirectory << "/" << fromDirectory
              << "' in the bucket '" << inputBucketName << "'." << std::endl;
    askQuestion("Enter return to start the copy.", alwaysTrueTest);

    if (!AwsDoc::Medical_Imaging::copySeriesBetweenBuckets(
            IDC_S3_BucketName,
            fromDirectory,
            inputBucketName,
            inputDirectory, clientConfiguration)) {
        std::cerr << "This workflow will exit because of an error." << std::endl;
        cleanup(stackName, dataStoreId, clientConfiguration);
        return false;
    }
```

DICOM ファイルを Amazon S3 データストアのにインポートします。

```
bool AwsDoc::Medical_Imaging::startDicomImport(const Aws::String &dataStoreID,
                                               const Aws::String
 &inputBucketName,
                                               const Aws::String &inputDirectory,
                                               const Aws::String
 &outputBucketName,
                                               const Aws::String
 &outputDirectory,
                                               const Aws::String &roleArn,
                                               Aws::String &importJobId,
                                               const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    bool result = false;
    if (startDICOMImportJob(dataStoreID, inputBucketName, inputDirectory,
                            outputBucketName, outputDirectory, roleArn,
 importJobId,
```

```
                                clientConfiguration)) {
        std::cout << "DICOM import job started with job ID " << importJobId <<
 "."
                  << std::endl;
        result = waitImportJobCompleted(dataStoreID, importJobId,
 clientConfiguration);
        if (result) {
            std::cout << "DICOM import job completed." << std::endl;

        }
    }

    return result;
}

//! Routine which starts a HealthImaging import job.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param inputBucketName: The name of the Amazon S3 bucket containing the DICOM
 files.
  \param inputDirectory: The directory in the S3 bucket containing the DICOM
 files.
  \param outputBucketName: The name of the S3 bucket for the output.
  \param outputDirectory: The directory in the S3 bucket to store the output.
  \param roleArn: The ARN of the IAM role with permissions for the import.
  \param importJobId: A string to receive the import job ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::startDICOMImportJob(
        const Aws::String &dataStoreID, const Aws::String &inputBucketName,
        const Aws::String &inputDirectory, const Aws::String &outputBucketName,
        const Aws::String &outputDirectory, const Aws::String &roleArn,
        Aws::String &importJobId,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(clientConfig);
    Aws::String inputURI = "s3://" + inputBucketName + "/" + inputDirectory +
 "/";
    Aws::String outputURI = "s3://" + outputBucketName + "/" + outputDirectory +
 "/";
    Aws::MedicalImaging::Model::StartDICOMImportJobRequest
 startDICOMImportJobRequest;
    startDICOMImportJobRequest.SetDatastoreId(dataStoreID);
    startDICOMImportJobRequest.SetDataAccessRoleArn(roleArn);
```

```
    startDICOMImportJobRequest.SetInputS3Uri(inputURI);
    startDICOMImportJobRequest.SetOutputS3Uri(outputURI);

    Aws::MedicalImaging::Model::StartDICOMImportJobOutcome
 startDICOMImportJobOutcome = medicalImagingClient.StartDICOMImportJob(
            startDICOMImportJobRequest);

    if (startDICOMImportJobOutcome.IsSuccess()) {
        importJobId = startDICOMImportJobOutcome.GetResult().GetJobId();
    }
    else {
        std::cerr << "Failed to start DICOM import job because "
                  << startDICOMImportJobOutcome.GetError().GetMessage() <<
 std::endl;
    }

    return startDICOMImportJobOutcome.IsSuccess();
}


//! Routine which waits for a DICOM import job to complete.
/*!
 * @param dataStoreID: The HealthImaging data store ID.
 * @param importJobId: The import job ID.
 * @param clientConfiguration : Aws client configuration.
 * @return  bool: Function succeeded.
 */
bool AwsDoc::Medical_Imaging::waitImportJobCompleted(const Aws::String
 &datastoreID,
                                                     const Aws::String
 &importJobId,
                                                     const
 Aws::Client::ClientConfiguration &clientConfiguration) {

    Aws::MedicalImaging::Model::JobStatus jobStatus =
 Aws::MedicalImaging::Model::JobStatus::IN_PROGRESS;
    while (jobStatus == Aws::MedicalImaging::Model::JobStatus::IN_PROGRESS) {
        std::this_thread::sleep_for(std::chrono::seconds(1));

        Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
 getDicomImportJobOutcome = getDICOMImportJob(
                datastoreID, importJobId,
                clientConfiguration);
```

```
        if (getDicomImportJobOutcome.IsSuccess()) {
            jobStatus =
 getDicomImportJobOutcome.GetResult().GetJobProperties().GetJobStatus();

            std::cout << "DICOM import job status: " <<

 Aws::MedicalImaging::Model::JobStatusMapper::GetNameForJobStatus(
                              jobStatus) << std::endl;
        }
        else {
            std::cerr << "Failed to get import job status because "
                      << getDicomImportJobOutcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    }

    return jobStatus == Aws::MedicalImaging::Model::JobStatus::COMPLETED;
}

//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
  \param clientConfig: Aws client configuration.
  \return GetDICOMImportJobOutcome: The import job outcome.
*/
Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
AwsDoc::Medical_Imaging::getDICOMImportJob(const Aws::String &dataStoreID,
                                           const Aws::String &importJobID,
                                           const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetDICOMImportJobRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetJobId(importJobID);
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome outcome =
 client.GetDICOMImportJob(
            request);
    if (!outcome.IsSuccess()) {
        std::cerr << "GetDICOMImportJob error: "
                  << outcome.GetError().GetMessage() << std::endl;
    }
```

```
    return outcome;
}
```

DICOM インポートジョブによって作成された画像セットを取得します。

```
bool
AwsDoc::Medical_Imaging::getImageSetsForDicomImportJob(const Aws::String
 &datastoreID,
                                                       const Aws::String
 &importJobId,
                                                       Aws::Vector<Aws::String>
 &imageSets,
                                                       const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome getDicomImportJobOutcome
 = getDICOMImportJob(
            datastoreID, importJobId, clientConfiguration);
    bool result = false;
    if (getDicomImportJobOutcome.IsSuccess()) {
        auto outputURI =
 getDicomImportJobOutcome.GetResult().GetJobProperties().GetOutputS3Uri();
        Aws::Http::URI uri(outputURI);
        const Aws::String &bucket = uri.GetAuthority();
        Aws::String key = uri.GetPath();

        Aws::S3::S3Client s3Client(clientConfiguration);
        Aws::S3::Model::GetObjectRequest objectRequest;
        objectRequest.SetBucket(bucket);
        objectRequest.SetKey(key + "/" + IMPORT_JOB_MANIFEST_FILE_NAME);

        auto getObjectOutcome = s3Client.GetObject(objectRequest);
        if (getObjectOutcome.IsSuccess()) {
            auto &data = getObjectOutcome.GetResult().GetBody();

            std::stringstream stringStream;
            stringStream << data.rdbuf();

            try {
                // Use JMESPath to extract the image set IDs.
                // https://jmespath.org/specification.html
                std::string jmesPathExpression =
 "jobSummary.imageSetsSummary[].imageSetId";
```

```
                jsoncons::json doc = jsoncons::json::parse(stringStream.str());

                jsoncons::json imageSetsJson = jsoncons::jmespath::search(doc,

 jmesPathExpression);\
                for (auto &imageSet: imageSetsJson.array_range()) {
                    imageSets.push_back(imageSet.as_string());
                }

                result = true;
            }
            catch (const std::exception &e) {
                std::cerr << e.what() << '\n';
            }

        }
        else {
            std::cerr << "Failed to get object because "
                      << getObjectOutcome.GetError().GetMessage() << std::endl;
        }

    }
    else {
        std::cerr << "Failed to get import job status because "
                  << getDicomImportJobOutcome.GetError().GetMessage() <<
 std::endl;
    }

    return result;
}
```

画像セットの画像フレーム情報を取得します。

```
bool AwsDoc::Medical_Imaging::getImageFramesForImageSet(const Aws::String
 &dataStoreID,
                                                        const Aws::String
 &imageSetID,
                                                        const Aws::String
 &outDirectory,

 Aws::Vector<ImageFrameInfo> &imageFrames,
```

```
                                                        const
Aws::Client::ClientConfiguration &clientConfiguration) {
   Aws::String fileName = outDirectory + "/" + imageSetID +
"_metadata.json.gzip";
   bool result = false;
   if (getImageSetMetadata(dataStoreID, imageSetID, "", // Empty string for
version ID.
                           fileName, clientConfiguration)) {
       try {
           std::string metadataGZip;
           {
               std::ifstream inFileStream(fileName.c_str(), std::ios::binary);
               if (!inFileStream) {
                   throw std::runtime_error("Failed to open file " + fileName);
               }

               std::stringstream stringStream;
               stringStream << inFileStream.rdbuf();
               metadataGZip = stringStream.str();
           }
           std::string metadataJson = gzip::decompress(metadataGZip.data(),
                                                       metadataGZip.size());
           // Use JMESPath to extract the image set IDs.
           // https://jmespath.org/specification.html
           jsoncons::json doc = jsoncons::json::parse(metadataJson);
           std::string jmesPathExpression = "Study.Series.*.Instances[].*[]";
           jsoncons::json instances = jsoncons::jmespath::search(doc,

jmesPathExpression);
           for (auto &instance: instances.array_range()) {
               jmesPathExpression = "DICOM.RescaleSlope";
               std::string rescaleSlope = jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();
               jmesPathExpression = "DICOM.RescaleIntercept";
               std::string rescaleIntercept =
jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();

               jmesPathExpression = "ImageFrames[][]";
               jsoncons::json imageFramesJson =
jsoncons::jmespath::search(instance,
```

```
 jmesPathExpression);

                for (auto &imageFrame: imageFramesJson.array_range()) {
                    ImageFrameInfo imageFrameIDs;
                    imageFrameIDs.mImageSetId = imageSetID;
                    imageFrameIDs.mImageFrameId = imageFrame.find(
                            "ID")->value().as_string();
                    imageFrameIDs.mRescaleIntercept = rescaleIntercept;
                    imageFrameIDs.mRescaleSlope = rescaleSlope;
                    imageFrameIDs.MinPixelValue = imageFrame.find(
                            "MinPixelValue")->value().as_string();
                    imageFrameIDs.MaxPixelValue = imageFrame.find(
                            "MaxPixelValue")->value().as_string();

                    jmesPathExpression =
 "max_by(PixelDataChecksumFromBaseToFullResolution, &Width).Checksum";
                    jsoncons::json checksumJson =
 jsoncons::jmespath::search(imageFrame,

 jmesPathExpression);
                    imageFrameIDs.mFullResolutionChecksum =
 checksumJson.as_integer<uint32_t>();

                    imageFrames.emplace_back(imageFrameIDs);
                }
            }

            result = true;
        }
        catch (const std::exception &e) {
            std::cerr << "getImageFramesForImageSet failed because " << e.what()
                      << std::endl;
        }
    }

    return result;
}

//! Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The HealthImaging image set ID.
  \param versionID: The HealthImaging image set version ID, ignored if empty.
```

```
  \param outputFilePath: The path where the metadata will be stored as gzipped
 json.
  \param clientConfig: Aws client configuration.
  \\return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
 &outputFilePath,
                                                  const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    if (!versionID.empty()) {
        request.SetVersionId(versionID);
    }
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
 client.GetImageSetMetadata(
            request);
    if (outcome.IsSuccess()) {
        std::ofstream file(outputFilePath, std::ios::binary);
        auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
        file << metadata.rdbuf();
    }
    else {
        std::cerr << "Failed to get image set metadata: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

イメージフレームをダウンロード、デコード、検証します。

```
bool AwsDoc::Medical_Imaging::downloadDecodeAndCheckImageFrames(
        const Aws::String &dataStoreID,
        const Aws::Vector<ImageFrameInfo> &imageFrames,
        const Aws::String &outDirectory,
        const Aws::Client::ClientConfiguration &clientConfiguration) {
```

```
   Aws::Client::ClientConfiguration clientConfiguration1(clientConfiguration);
   clientConfiguration1.executor =
Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>(
           "executor", 25);
   Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(
           clientConfiguration1);

   Aws::Utils::Threading::Semaphore semaphore(0, 1);
   std::atomic<size_t> count(imageFrames.size());

   bool result = true;
   for (auto &imageFrame: imageFrames) {
       Aws::MedicalImaging::Model::GetImageFrameRequest getImageFrameRequest;
       getImageFrameRequest.SetDatastoreId(dataStoreID);
       getImageFrameRequest.SetImageSetId(imageFrame.mImageSetId);

       Aws::MedicalImaging::Model::ImageFrameInformation imageFrameInformation;
       imageFrameInformation.SetImageFrameId(imageFrame.mImageFrameId);
       getImageFrameRequest.SetImageFrameInformation(imageFrameInformation);

       auto getImageFrameAsyncLambda = [&semaphore, &result, &count, imageFrame,
outDirectory](
               const Aws::MedicalImaging::MedicalImagingClient *client,
               const Aws::MedicalImaging::Model::GetImageFrameRequest &request,
               Aws::MedicalImaging::Model::GetImageFrameOutcome outcome,
               const std::shared_ptr<const Aws::Client::AsyncCallerContext>
&context) {

               if (!handleGetImageFrameResult(outcome, outDirectory,
imageFrame)) {
                   std::cerr << "Failed to download and convert image frame: "
                             << imageFrame.mImageFrameId << " from image set: "
                             << imageFrame.mImageSetId << std::endl;
                   result = false;
               }

               count--;
               if (count <= 0) {

                   semaphore.ReleaseAll();
               }
       }; // End of 'getImageFrameAsyncLambda' lambda.
```

```
        medicalImagingClient.GetImageFrameAsync(getImageFrameRequest,
                                                getImageFrameAsyncLambda);
    }

    if (count > 0) {
        semaphore.WaitOne();
    }

    if (result) {
        std::cout << imageFrames.size() << " image files were downloaded."
                  << std::endl;
    }

    return result;
}

bool AwsDoc::Medical_Imaging::decodeJPHFileAndValidateWithChecksum(
        const Aws::String &jphFile,
        uint32_t crc32Checksum) {
    opj_image_t *outputImage = jphImageToOpjBitmap(jphFile);
    if (!outputImage) {
        return false;
    }

    bool result = true;
    if (!verifyChecksumForImage(outputImage, crc32Checksum)) {
        std::cerr << "The checksum for the image does not match the expected
 value."
                  << std::endl;
        std::cerr << "File :" << jphFile << std::endl;
        result = false;
    }

    opj_image_destroy(outputImage);

    return result;
}

opj_image *
AwsDoc::Medical_Imaging::jphImageToOpjBitmap(const Aws::String &jphFile) {
    opj_stream_t *inFileStream = nullptr;
    opj_codec_t *decompressorCodec = nullptr;
    opj_image_t *outputImage = nullptr;
    try {
```

```
        std::shared_ptr<opj_dparameters> decodeParameters =
std::make_shared<opj_dparameters>();
        memset(decodeParameters.get(), 0, sizeof(opj_dparameters));

        opj_set_default_decoder_parameters(decodeParameters.get());

        decodeParameters->decod_format = 1; // JP2 image format.
        decodeParameters->cod_format = 2; // BMP image format.

        std::strncpy(decodeParameters->infile, jphFile.c_str(),
                     OPJ_PATH_LEN);

        inFileStream = opj_stream_create_default_file_stream(
                decodeParameters->infile, true);
        if (!inFileStream) {
            throw std::runtime_error(
                    "Unable to create input file stream for file '" + jphFile +
"'.");
        }

        decompressorCodec = opj_create_decompress(OPJ_CODEC_JP2);
        if (!decompressorCodec) {
            throw std::runtime_error("Failed to create decompression codec.");
        }

        int decodeMessageLevel = 1;
        if (!setupCodecLogging(decompressorCodec, &decodeMessageLevel)) {
            std::cerr << "Failed to setup codec logging." << std::endl;
        }

        if (!opj_setup_decoder(decompressorCodec, decodeParameters.get())) {
            throw std::runtime_error("Failed to setup decompression codec.");
        }
        if (!opj_codec_set_threads(decompressorCodec, 4)) {
            throw std::runtime_error("Failed to set decompression codec
threads.");
        }

        if (!opj_read_header(inFileStream, decompressorCodec, &outputImage)) {
            throw std::runtime_error("Failed to read header.");
        }

        if (!opj_decode(decompressorCodec, inFileStream,
                        outputImage)) {
```

```
            throw std::runtime_error("Failed to decode.");
        }

        if (DEBUGGING) {
            std::cout << "image width : " << outputImage->x1 - outputImage->x0
                      << std::endl;
            std::cout << "image height : " << outputImage->y1 - outputImage->y0
                      << std::endl;
            std::cout << "number of channels: " << outputImage->numcomps
                      << std::endl;
            std::cout << "colorspace : " << outputImage->color_space <<
 std::endl;
        }

    } catch (const std::exception &e) {
        std::cerr << e.what() << std::endl;
        if (outputImage) {
            opj_image_destroy(outputImage);
            outputImage = nullptr;
        }
    }
    if (inFileStream) {
        opj_stream_destroy(inFileStream);
    }
    if (decompressorCodec) {
        opj_destroy_codec(decompressorCodec);
    }

    return outputImage;
}

//! Template function which converts a planar image bitmap to an interleaved
 image bitmap and
//! then verifies the checksum of the bitmap.
/*!
 * @param image: The OpenJPEG image struct.
 * @param crc32Checksum: The CRC32 checksum.
 * @return  bool: Function succeeded.
 */
template<class myType>
bool verifyChecksumForImageForType(opj_image_t *image, uint32_t crc32Checksum) {
    uint32_t width = image->x1 - image->x0;
    uint32_t height = image->y1 - image->y0;
    uint32_t numOfChannels = image->numcomps;
```

```
    // Buffer for interleaved bitmap.
    std::vector<myType> buffer(width * height * numOfChannels);

    // Convert planar bitmap to interleaved bitmap.
    for (uint32_t channel = 0; channel < numOfChannels; channel++) {
        for (uint32_t row = 0; row < height; row++) {
            uint32_t fromRowStart = row / image->comps[channel].dy * width /
                                    image->comps[channel].dx;
            uint32_t toIndex = (row * width) * numOfChannels + channel;

            for (uint32_t col = 0; col < width; col++) {
                uint32_t fromIndex = fromRowStart + col / image-
>comps[channel].dx;

                buffer[toIndex] = static_cast<myType>(image-
>comps[channel].data[fromIndex]);

                toIndex += numOfChannels;
            }
        }
    }

    // Verify checksum.
    boost::crc_32_type crc32;
    crc32.process_bytes(reinterpret_cast<char *>(buffer.data()),
                        buffer.size() * sizeof(myType));

    bool result = crc32.checksum() == crc32Checksum;
    if (!result) {
        std::cerr << "verifyChecksumForImage, checksum mismatch, expected - "
                  << crc32Checksum << ", actual - " << crc32.checksum()
                  << std::endl;
    }

    return result;
}

//! Routine which verifies the checksum of an OpenJPEG image struct.
/*!
 * @param image: The OpenJPEG image struct.
 * @param crc32Checksum: The CRC32 checksum.
 * @return  bool: Function succeeded.
 */
```

```
bool AwsDoc::Medical_Imaging::verifyChecksumForImage(opj_image_t *image,
                                                     uint32_t crc32Checksum) {
    uint32_t channels = image->numcomps;
    bool result = false;
    if (0 < channels) {
        // Assume the precision is the same for all channels.
        uint32_t precision = image->comps[0].prec;
        bool signedData = image->comps[0].sgnd;
        uint32_t bytes = (precision + 7) / 8;

        if (signedData) {
            switch (bytes) {
                case 1 :
                    result = verifyChecksumForImageForType<int8_t>(image,

 crc32Checksum);
                    break;
                case 2 :
                    result = verifyChecksumForImageForType<int16_t>(image,

 crc32Checksum);
                    break;
                case 4 :
                    result = verifyChecksumForImageForType<int32_t>(image,

 crc32Checksum);
                    break;
                default:
                    std::cerr
                            << "verifyChecksumForImage, unsupported data type,
 signed bytes - "
                            << bytes << std::endl;
                    break;
            }
        }
        else {
            switch (bytes) {
                case 1 :
                    result = verifyChecksumForImageForType<uint8_t>(image,

 crc32Checksum);
                    break;
                case 2 :
                    result = verifyChecksumForImageForType<uint16_t>(image,
```

```
 crc32Checksum);
                    break;
                case 4 :
                    result = verifyChecksumForImageForType<uint32_t>(image,

 crc32Checksum);
                    break;
                default:
                    std::cerr
                            << "verifyChecksumForImage, unsupported data type,
 unsigned bytes - "
                            << bytes << std::endl;
                    break;
            }
        }

        if (!result) {
            std::cerr << "verifyChecksumForImage, error bytes " << bytes
                      << " signed "
                      << signedData << std::endl;
        }
    }
    else {
        std::cerr << "'verifyChecksumForImage', no channels in the image."
                  << std::endl;
    }
    return result;
}
```

リソースをクリーンアップします。

```
bool AwsDoc::Medical_Imaging::cleanup(const Aws::String &stackName,
                                      const Aws::String &dataStoreId,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    bool result = true;

    if (!stackName.empty() && askYesNoQuestion(
            "Would you like to delete the stack " + stackName + "? (y/n)")) {
        std::cout << "Deleting the image sets in the stack." << std::endl;
        result &= emptyDatastore(dataStoreId, clientConfiguration);
```

```
        printAsterisksLine();
        std::cout << "Deleting the stack." << std::endl;
        result &= deleteStack(stackName, clientConfiguration);
    }
    return result;
}

bool AwsDoc::Medical_Imaging::emptyDatastore(const Aws::String &datastoreID,
                                             const
 Aws::Client::ClientConfiguration &clientConfiguration) {

    Aws::MedicalImaging::Model::SearchCriteria emptyCriteria;
    Aws::Vector<Aws::String> imageSetIDs;
    bool result = false;
    if (searchImageSets(datastoreID, emptyCriteria, imageSetIDs,
                        clientConfiguration)) {
        result = true;
        for (auto &imageSetID: imageSetIDs) {
            result &= deleteImageSet(datastoreID, imageSetID,
 clientConfiguration);
        }
    }

    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の以下のトピックを参照してください。
  - DeleteImageSet
  - GetDICOMImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - StartDICOMImportJob

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

index.js - ステップをオーケストレーションします。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
```

```
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
```

```
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js - リソースをデプロイします。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
```

```
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
```

```
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName == state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
```

```
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

dataset-steps.js - DICOM ファイルをコピーします。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  S3Client,
```

```
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
```

```
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
```

```
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js - データストアへのインポートを開始します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
```

```
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
```

```
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js - 画像セット IDs。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
```

```
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js - イメージフレーム IDs。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
```

```
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });
```

```
      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
  { slow: false },
);
```

verify-steps.js - イメージフレームを確認します。AWS HealthImaging Pixel Data Verification ライブラリが検証に使用されました。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
```

```
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (/** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
```

```
      const datastoreId = state.stackOutputs.DatastoreID;
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid}
 and sop ${sopInstanceUid}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(
                    `Verification tool exited with code ${code} for image set
 ${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

clean-up-steps.js - リソースを破棄します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
```

```
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
```

```
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - DeleteImageSet
  - GetDICOMImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - StartDICOMImportJob

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

必要なリソースを持つ AWS CloudFormation スタックを作成します。

```
    def deploy(self):
        """
        Deploys prerequisite resources used by the scenario. The resources are
        defined in the associated `setup.yaml` AWS CloudFormation script and are
deployed
        as a CloudFormation stack, so they can be easily managed and destroyed.
        """

        print("\t\tLet's deploy the stack for resource creation.")
        stack_name = q.ask("\t\tEnter a name for the stack: ", q.non_empty)

        data_store_name = q.ask(
            "\t\tEnter a name for the Health Imaging Data Store: ", q.non_empty
        )

        account_id = boto3.client("sts").get_caller_identity()["Account"]

        with open(
```

```
            "../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml"
        ) as setup_file:
            setup_template = setup_file.read()
        print(f"\t\tCreating {stack_name}.")
        stack = self.cf_resource.create_stack(
            StackName=stack_name,
            TemplateBody=setup_template,
            Capabilities=["CAPABILITY_NAMED_IAM"],
            Parameters=[
                {
                    "ParameterKey": "datastoreName",
                    "ParameterValue": data_store_name,
                },
                {
                    "ParameterKey": "userAccountID",
                    "ParameterValue": account_id,
                },
            ],
        )
        print("\t\tWaiting for stack to deploy. This typically takes a minute or
 two.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
        waiter.wait(StackName=stack.name)
        stack.load()
        print(f"\t\tStack status: {stack.stack_status}")

        outputs_dictionary = {
            output["OutputKey"]: output["OutputValue"] for output in
 stack.outputs
        }
        self.input_bucket_name = outputs_dictionary["BucketName"]
        self.output_bucket_name = outputs_dictionary["BucketName"]
        self.role_arn = outputs_dictionary["RoleArn"]
        self.data_store_id = outputs_dictionary["DatastoreID"]
        return stack
```

DICOM ファイルを Amazon S3 インポートバケットにコピーします。

```
    def copy_single_object(self, key, source_bucket, target_bucket,
 target_directory):
```

```
        """
        Copies a single object from a source to a target bucket.

        :param key: The key of the object to copy.
        :param source_bucket: The source bucket for the copy.
        :param target_bucket: The target bucket for the copy.
        :param target_directory: The target directory for the copy.
        """
        new_key = target_directory + "/" + key
        copy_source = {"Bucket": source_bucket, "Key": key}
        self.s3_client.copy_object(
            CopySource=copy_source, Bucket=target_bucket, Key=new_key
        )
        print(f"\n\t\tCopying {key}.")

    def copy_images(
        self, source_bucket, source_directory, target_bucket, target_directory
    ):
        """
        Copies the images from the source to the target bucket using multiple
threads.

        :param source_bucket: The source bucket for the images.
        :param source_directory: Directory within the source bucket.
        :param target_bucket: The target bucket for the images.
        :param target_directory: Directory within the target bucket.
        """

        # Get list of all objects in source bucket.
        list_response = self.s3_client.list_objects_v2(
            Bucket=source_bucket, Prefix=source_directory
        )
        objs = list_response["Contents"]
        keys = [obj["Key"] for obj in objs]

        # Copy the objects in the bucket.
        for key in keys:
            self.copy_single_object(key, source_bucket, target_bucket,
target_directory)

        print("\t\tDone copying all objects.")
```

DICOM ファイルを Amazon S3 データストアのにインポートします。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def start_dicom_import_job(
        self,
        data_store_id,
        input_bucket_name,
        input_directory,
        output_bucket_name,
        output_directory,
        role_arn,
    ):
        """
        Routine which starts a HealthImaging import job.

        :param data_store_id: The HealthImaging data store ID.
        :param input_bucket_name: The name of the Amazon S3 bucket containing the
 DICOM files.
        :param input_directory: The directory in the S3 bucket containing the
 DICOM files.
        :param output_bucket_name: The name of the S3 bucket for the output.
        :param output_directory: The directory in the S3 bucket to store the
 output.
        :param role_arn: The ARN of the IAM role with permissions for the import.
        :return: The job ID of the import.
```

```
        """

        input_uri = f"s3://{input_bucket_name}/{input_directory}/"
        output_uri = f"s3://{output_bucket_name}/{output_directory}/"
        try:
            job = self.medical_imaging_client.start_dicom_import_job(
                jobName="examplejob",
                datastoreId=data_store_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_uri,
                outputS3Uri=output_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

DICOM インポートジョブによって作成された画像セットを取得します。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
```

```
        return cls(medical_imaging_client, s3_client)


    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
        """
        Retrieves the image sets created for an import job.

        :param datastore_id: The HealthImaging data store ID
        :param import_job_id: The import job ID
        :return: List of image set IDs
        """

        import_job = self.medical_imaging_client.get_dicom_import_job(
            datastoreId=datastore_id, jobId=import_job_id
        )

        output_uri = import_job["jobProperties"]["outputS3Uri"]

        bucket = output_uri.split("/")[2]
        key = "/".join(output_uri.split("/")[3:])

        # Try to get the manifest.
        retries = 3
        while retries > 0:
            try:
                obj = self.s3_client.get_object(
                    Bucket=bucket, Key=key + "job-output-manifest.json"
                )
                body = obj["Body"]
                break
            except ClientError as error:
                retries = retries - 1
                time.sleep(3)
        try:
            data = json.load(body)
            expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
            image_sets = expression.search(data)
        except json.decoder.JSONDecodeError as error:
            image_sets = import_job["jobProperties"]

        return image_sets
```

```
    def get_image_set(self, datastore_id, image_set_id, version_id=None):
        """
        Get the properties of an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.medical_imaging_client.get_image_set(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set = self.medical_imaging_client.get_image_set(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
        except ClientError as err:
            logger.error(
                "Couldn't get image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return image_set
```

画像セットの画像フレーム情報を取得します。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
```

```
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_frames_for_image_set(self, datastore_id, image_set_id,
out_directory):
        """
        Get the image frames for an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param out_directory: The directory to save the file.
        :return: The image frames.
        """
        image_frames = []
        file_name = os.path.join(out_directory,
f"{image_set_id}_metadata.json.gzip")
        file_name = file_name.replace("/", "\\\\")
        self.get_image_set_metadata(file_name, datastore_id, image_set_id)
        try:
            with gzip.open(file_name, "rb") as f_in:
                doc = json.load(f_in)
            instances = jmespath.search("Study.Series.*.Instances[].*[]", doc)
            for instance in instances:
                rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
                rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)
                image_frames_json = jmespath.search("ImageFrames[][]", instance)
                for image_frame in image_frames_json:
                    checksum_json = jmespath.search(
                        "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",
                        image_frame,
                    )
                    image_frame_info = {
                        "imageSetId": image_set_id,
                        "imageFrameId": image_frame["ID"],
```

```
                        "rescaleIntercept": rescale_intercept,
                        "rescaleSlope": rescale_slope,
                        "minPixelValue": image_frame["MinPixelValue"],
                        "maxPixelValue": image_frame["MaxPixelValue"],
                        "fullResolutionChecksum": checksum_json["Checksum"],
                    }
                    image_frames.append(image_frame_info)
            return image_frames
        except TypeError:
            return {}
        except ClientError as err:
            logger.error(
                "Couldn't get image frames for image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        return image_frames


    def get_image_set_metadata(
        self, metadata_file, datastore_id, image_set_id, version_id=None
    ):
        """
        Get the metadata of an image set.

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """

        try:
            if version_id:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
```

```
                )
            with open(metadata_file, "wb") as f:
                for chunk in
 image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)

        except ClientError as err:
            logger.error(
                "Couldn't get image metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

イメージフレームをダウンロード、デコード、検証します。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_pixel_data(
        self, file_path_to_write, datastore_id, image_set_id, image_frame_id
    ):
        """
```

```
        Get an image frame's pixel data.

        :param file_path_to_write: The path to write the image frame's HTJ2K
 encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.medical_imaging_client.get_image_frame(
                datastoreId=datastore_id,
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    f.write(chunk)
        except ClientError as err:
            logger.error(
                "Couldn't get image frame. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def download_decode_and_check_image_frames(
        self, data_store_id, image_frames, out_directory
    ):
        """
        Downloads image frames, decodes them, and uses the checksum to validate
        the decoded images.

        :param data_store_id: The HealthImaging data store ID.
        :param image_frames: A list of dicts containing image frame information.
        :param out_directory: A directory for the downloaded images.
        :return: True if the function succeeded; otherwise, False.
        """
        total_result = True
        for image_frame in image_frames:
            image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
            self.get_pixel_data(
                image_file_path,
```

```
                data_store_id,
                image_frame["imageSetId"],
                image_frame["imageFrameId"],
            )

            image_array = self.jph_image_to_opj_bitmap(image_file_path)
            crc32_checksum = image_frame["fullResolutionChecksum"]
            # Verify checksum.
            crc32_calculated = zlib.crc32(image_array)
            image_result = crc32_checksum == crc32_calculated
            print(
                f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
{image_result }"
            )
            total_result = total_result and image_result
        return total_result

    @staticmethod
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array
```

リソースをクリーンアップします。

```
    def destroy(self, stack):
        """
        Destroys the resources managed by the CloudFormation stack, and the
CloudFormation
        stack itself.
```

```
        :param stack: The CloudFormation stack that manages the example
 resources.
        """

        print(f"\t\tCleaning up resources and {stack.name}.")
        data_store_id = None
        for oput in stack.outputs:
            if oput["OutputKey"] == "DatastoreID":
                data_store_id = oput["OutputValue"]
        if data_store_id is not None:
            print(f"\t\tDeleting image sets in data store {data_store_id}.")
            image_sets = self.medical_imaging_wrapper.search_image_sets(
                data_store_id, {}
            )
            image_set_ids = [image_set["imageSetId"] for image_set in image_sets]

            for image_set_id in image_set_ids:
                self.medical_imaging_wrapper.delete_image_set(
                    data_store_id, image_set_id
                )
                print(f"\t\tDeleted image set with id : {image_set_id}")

        print(f"\t\tDeleting {stack.name}.")
        stack.delete()
        print("\t\tWaiting for stack removal. This may take a few minutes.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_delete_complete")
        waiter.wait(StackName=stack.name)
        print("\t\tStack delete complete.")




class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
```

```
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.medical_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries


    def delete_image_set(self, datastore_id, image_set_id):
        """
        Delete an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        """
        try:
```

```
            delete_results = self.medical_imaging_client.delete_image_set(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                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。
  - DeleteImageSet
  - GetDICOMImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - StartDICOMImportJob

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 HealthImaging データストアのタグ付け

次のコード例は、 HealthImaging データストアにタグを付ける方法を示しています。

Java

SDK for Java 2.x

データストアにタグを付けます。

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

リソースにタグを付けるためのユーティリティ関数。

```
   public static void tagMedicalImagingResource(MedicalImagingClient
 medicalImagingClient,
           String resourceArn,
           Map<String, String> tags) {
       try {
           TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tags(tags)
                   .build();

           medicalImagingClient.tagResource(tagResourceRequest);

           System.out.println("Tags have been added to the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

データストアのタグを一覧表示します。

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                ListTagsForResourceResponse result =
 ListTagsForResource.listMedicalImagingResourceTags(
```

```
                                medicalImagingClient,
                                datastoreArn);
                if (result != null) {
                        System.out.println("Tags for resource: " +
 result.tags());
                }
```

リソースのタグを一覧表示するユーティリティ関数。

```
   public static ListTagsForResourceResponse
 listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
            String resourceArn) {
        try {
            ListTagsForResourceRequest listTagsForResourceRequest =
 ListTagsForResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .build();

            return
 medicalImagingClient.listTagsForResource(listTagsForResourceRequest);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

データストアのタグを解除するには

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

リソースのタグを解除するユーティリティ関数。

```
   public static void untagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Collection<String> tagKeys) {
       try {
           UntagResourceRequest untagResourceRequest =
UntagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tagKeys(tagKeys)
                   .build();

           medicalImagingClient.untagResource(untagResourceRequest);

           System.out.println("Tags have been removed from the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

データストアにタグを付けます。

```
try {
```

```
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(datastoreArn, tags);
  } catch (e) {
    console.log(e);
  }
```

リソースにタグを付けるためのユーティリティ関数。

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }
```

```
  return response;
};
```

データストアのタグを一覧表示します。

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(datastoreArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

リソースのタグを一覧表示するユーティリティ関数。

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
```

```
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

データストアのタグを解除するには

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(datastoreArn, keys);
  } catch (e) {
    console.log(e);
  }
```

リソースのタグを解除するユーティリティ関数。

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
```

```
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     }
  // }

  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

データストアにタグを付けます。

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

リソースにタグを付けるためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
```

```
        self.health_imaging_client = health_imaging_client


    def tag_resource(self, resource_arn, tags):
        """
        Tag a resource.

        :param resource_arn: The ARN of the resource.
        :param tags: The tags to apply.
        """
        try:
            self.health_imaging_client.tag_resource(resourceArn=resource_arn,
 tags=tags)
        except ClientError as err:
            logger.error(
                "Couldn't tag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

データストアのタグを一覧表示します。

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

リソースのタグを一覧表示するユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_tags_for_resource(self, resource_arn):
        """
        List the tags for a resource.

        :param resource_arn: The ARN of the resource.
```

```
        :return: The list of tags.
        """
        try:
            tags = self.health_imaging_client.list_tags_for_resource(
                resourceArn=resource_arn
            )
        except ClientError as err:
            logger.error(
                "Couldn't list tags for resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tags["tags"]
```

データストアのタグを解除するには

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

リソースのタグを解除するユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def untag_resource(self, resource_arn, tag_keys):
        """
        Untag a resource.

        :param resource_arn: The ARN of the resource.
        :param tag_keys: The tag keys to remove.
        """
        try:
            self.health_imaging_client.untag_resource(
```

```
                resourceArn=resource_arn, tagKeys=tag_keys
            )
        except ClientError as err:
            logger.error(
                "Couldn't untag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは MedicalImagingWrapper 、 オブジェクトをインスタンス化します。

```
client = boto3.client("medical-imaging")
medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- 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。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 HealthImaging 画像セットのタグ付け

次のコード例は、 HealthImaging 画像セットにタグを付ける方法を示しています。

Java

SDK for Java 2.x

イメージセットにタグを付けるには

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

リソースにタグを付けるためのユーティリティ関数。

```
    public static void tagMedicalImagingResource(MedicalImagingClient
 medicalImagingClient,
            String resourceArn,
            Map<String, String> tags) {
        try {
            TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .tags(tags)
                    .build();

            medicalImagingClient.tagResource(tagResourceRequest);

            System.out.println("Tags have been added to the resource.");
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

イメージセットのタグを一覧表示します。

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
```

```
                ListTagsForResourceResponse result =
 ListTagsForResource.listMedicalImagingResourceTags(
                                medicalImagingClient,
                                imageSetArn);
                if (result != null) {
                        System.out.println("Tags for resource: " +
 result.tags());
                }
```

リソースのタグを一覧表示するユーティリティ関数。

```
   public static ListTagsForResourceResponse
 listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
           String resourceArn) {
       try {
           ListTagsForResourceRequest listTagsForResourceRequest =
 ListTagsForResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .build();

           return
 medicalImagingClient.listTagsForResource(listTagsForResourceRequest);
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return null;
   }
```

イメージセットのタグを解除します。

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                Collections.singletonList("Deployment"));
```

リソースのタグを解除するユーティリティ関数。

```
   public static void untagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Collection<String> tagKeys) {
       try {
           UntagResourceRequest untagResourceRequest =
UntagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tagKeys(tagKeys)
                   .build();

           medicalImagingClient.untagResource(untagResourceRequest);

           System.out.println("Tags have been removed from the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

イメージセットにタグを付けるには

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(imagesetArn, tags);
  } catch (e) {
    console.log(e);
  }
```

リソースにタグを付けるためのユーティリティ関数。

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
```

```
  // }

  return response;
};
```

イメージセットのタグを一覧表示します。

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(imagesetArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

リソースのタグを一覧表示するユーティリティ関数。

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
```

```
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

イメージセットのタグを解除します。

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(imagesetArn, keys);
  } catch (e) {
    console.log(e);
  }
```

リソースのタグを解除するユーティリティ関数。

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
```

```
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

イメージセットにタグを付けるには

```
    an_image_set_arn = (
        "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/"
        "imageset/12345678901234567890123456789012"
    )

    medical_imaging_wrapper.tag_resource(image_set_arn, {"Deployment":
 "Development"})
```

リソースにタグを付けるためのユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def tag_resource(self, resource_arn, tags):
        """
        Tag a resource.

        :param resource_arn: The ARN of the resource.
        :param tags: The tags to apply.
        """
        try:
            self.health_imaging_client.tag_resource(resourceArn=resource_arn,
 tags=tags)
        except ClientError as err:
            logger.error(
                "Couldn't tag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

イメージセットのタグを一覧表示します。

```
    an_image_set_arn = (
        "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/"
        "imageset/12345678901234567890123456789012"
    )

    medical_imaging_wrapper.list_tags_for_resource(image_set_arn)
```

リソースのタグを一覧表示するユーティリティ関数。

```
class MedicalImagingWrapper:
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def list_tags_for_resource(self, resource_arn):
        """
        List the tags for a resource.

        :param resource_arn: The ARN of the resource.
        :return: The list of tags.
        """
        try:
            tags = self.health_imaging_client.list_tags_for_resource(
                resourceArn=resource_arn
            )
        except ClientError as err:
            logger.error(
                "Couldn't list tags for resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tags["tags"]
```

イメージセットのタグを解除します。

```
    an_image_set_arn = (
        "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/"
        "imageset/12345678901234567890123456789012"
    )

    medical_imaging_wrapper.untag_resource(image_set_arn, ["Deployment"])
```

リソースのタグを解除するユーティリティ関数。

```
class MedicalImagingWrapper:
```

```
    def __init__(self, health_imaging_client):
        self.health_imaging_client = health_imaging_client


    def untag_resource(self, resource_arn, tag_keys):
        """
        Untag a resource.

        :param resource_arn: The ARN of the resource.
        :param tag_keys: The tag keys to remove.
        """
        try:
            self.health_imaging_client.untag_resource(
                resourceArn=resource_arn, tagKeys=tag_keys
            )
        except ClientError as err:
            logger.error(
                "Couldn't untag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

次のコードは、 MedicalImagingWrapper オブジェクトをインスタンス化します。

```
    client = boto3.client("medical-imaging")
    medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- 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。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

AWS SDK デベロッパー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 HealthImaging での 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 のモニタリング HealthImaging

モニタリングとログ記録は、AWS のセキュリティ、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 HealthImaging。 は、 を監視し HealthImaging、問題が発生した場合に報告し、必要に応じて自動アクションを実行するために、以下のログ記録とモニタリングツール AWS を提供します。

- AWS CloudTrail は、 AWS アカウントによって、またはアカウントに代わって行われた API コールおよび関連イベントをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。を呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、呼び出しが発生した日時を特定できます。詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。
- Amazon CloudWatch は、 AWS リソースと で実行しているアプリケーションを AWS リアルタイムでモニタリングします。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、 で Amazon EC2 インスタンスの CPU 使用率やその他のメトリクス CloudWatch を追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon ユーザーガイド CloudWatch 」を参照してください。
- Amazon EventBridge は、アプリケーションをさまざまなソースのデータに簡単に接続できるサーバーレスイベントバスサービスです。 は、独自のアプリケーション、S oftware-as-aサービス (SaaS) アプリケーション、および AWS のサービスからリアルタイムデータのストリームを EventBridge 配信し、そのデータを Lambda などのターゲットにルーティングします。これにより、サービスで発生したイベントをモニタリングし、イベント駆動型アーキテクチャを構築できます。詳細については、「Amazon ユーザーガイド EventBridge 」を参照してください。

トピック

AWS CloudTrail で を使用する HealthImaging

AWS HealthImaging は と統合されています。これは AWS CloudTrail、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 サービスです HealthImaging。 は、 の

すべての API コールをイベント HealthImaging として CloudTrail キャプチャします。キャプチャされた呼び出しには、 HealthImaging コンソールからの呼び出しと HealthImaging API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、 の CloudTrailイベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができます HealthImaging。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。によって収集された情報を使用して CloudTrail、 に対するリクエスト HealthImaging、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

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

証跡の作成

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

> **Note**
>
> HealthImaging で AWS の CloudTrail イベント履歴を表示するには AWS Management Console、ルックアップ属性メニューに移動し、イベントソース を選択し、 を選択しますmedical-imaging.amazonaws.com。

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

- 追跡を作成するための概要
- CloudTrail がサポートするサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

> **Note**
>
> AWS HealthImaging は、管理 CloudTrail イベントとデータイベントの 2 種類のイベントをサポートしています。管理イベントは、 を含むすべての AWS サービスが生成する一般的なイベントです HealthImaging。デフォルトでは、ログ記録は、有効になっているすべての HealthImaging API コールの管理イベントに適用されます。データイベントは課金可能で、通常は 1 秒あたりのトランザクション数 (tps) が高い APIs 用に予約されているため、コスト目的で CloudTrail ログの作成をオプトアウトできます。
> では HealthImaging、AWS API リファレンスに記載されているすべての HealthImaging APIアクションは、 を除く管理イベントと見なされます`GetImageFrame`。`GetImageFrame` アクションはデータイベント CloudTrail として にオンボードされるため、有効にする必要があります。詳細については、「AWS CloudTrail ユーザーガイド」の「データイベントをログ記録する」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 root または AWS Identity and Access Management (IAM) ユーザーの認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS 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。

ログエントリについて

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、 `GetDICOMImportJob`アクション HealthImaging を示す の CloudTrail ログエントリを示しています。

```
{
```

```
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "XXXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
        "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/
ce6d90ba-5fba-4456-a7bc-f9bc877597c3"
        "accountId": "123456789012",
        "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "XXXXXXXXXXXXXXXXXXXXX",
                "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
                "accountId": "123456789012",
                "userName": "TestAccessRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-28T15:52:42Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-28T16:02:30Z",
    "eventSource": "medical-imaging.amazonaws.com",
    "eventName": "GetDICOMImportJob",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-
Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync
 http/Apache cfg/retry-mode/standard",
    "requestParameters": {
        "jobId": "5d08d05d6aab2a27922d6260926077d4",
        "datastoreId": "12345678901234567890123456789012"
    },
    "responseElements": null,
    "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
    "eventID": "26307f73-07f4-4276-b379-d362aa303b22",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "824333766656",
    "eventCategory": "Management"
```

```
}
```

CloudWatch での Amazon の使用 HealthImaging

HealthImaging を使用して AWS をモニタリングできます。これにより CloudWatch、raw データを収集し、読み取り可能なほぼリアルタイムのメトリクスに加工できます。これらの統計は 15 か月間保持されるため、その履歴情報を利用して、ウェブアプリケーションまたはサービスの動作をより的確に把握できます。また、特定のしきい値をモニタリングするアラームを設定し、しきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon ユーザーガイド CloudWatch 」を参照してください。

> **Note**
>
> メトリクスは、すべての HealthImaging APIs。

次の表に、 のメトリクスとディメンションを示します HealthImaging。それぞれは、ユーザーが指定したデータ範囲の頻度カウントとして表示されます。

メトリクス

メトリクス	説明
コールカウント	API の呼び出し回数。これはアカウントまたは指定したデータストアについて報告できます。 単位: カウント 有効な統計: Sum、Count ディメンション: オペレーション、データストア ID、データストアタイプ

のメトリクスは、 AWS Management Console、、 AWS CLIまたは CloudWatch API HealthImaging を使用して取得できます。 CloudWatch API は、Amazon AWS Software Development Kit (SDKs または CloudWatch API ツールのいずれかから使用できます。 HealthImaging コンソールには、CloudWatch API の raw データに基づくグラフが表示されます。

HealthImaging でモニタリングするには、適切な CloudWatch アクセス許可が必要です CloudWatch。詳細については、「 ユーザーガイド」の「 の Identity and Access Management CloudWatchCloudWatch 」を参照してください。

HealthImaging メトリクスの表示

メトリクスを表示するには (CloudWatch コンソール)

1. にサインイン AWS Management Console し、CloudWatch コンソール を開きます。
2. [メトリクス] で、[すべてのメトリクス]、[AWS/Medical Imaging] の順に選択します。
3. ディメンションを選択してメトリクスの名前を選んだら、グラフに追加 を選択します。
4. 日付範囲の値を選択します。選択した日付範囲のメトリクスカウントがグラフに表示されます。

を使用したアラームの作成 CloudWatch

CloudWatch アラームは、指定した期間にわたって単一のメトリクスを監視し、Amazon Simple Notification Service (Amazon SNS) トピックまたは Auto Scaling ポリシーに通知を送信するという 1 つ以上のアクションを実行します。 Auto Scaling アクションは、指定した複数の期間における特定のしきい値に対するメトリクスの値に基づいています。 は、アラームの状態が変更されたときに Amazon SNS メッセージを送信 CloudWatch することもできます。

CloudWatch アラームは、状態が変化し、指定した期間持続した場合にのみアクションを呼び出します。詳細については、「アラームの使用 CloudWatch 」を参照してください。

EventBridge での Amazon の使用 HealthImaging

Amazon EventBridge は、イベントを使用してアプリケーションコンポーネントを接続できるサーバーレスサービスです。これにより、スケーラブルなイベント駆動型アプリケーションを簡単に構築できます。の基礎 EventBridge は、イベントをターゲット にルーティングするルールを作成することです。AWS HealthImaging は、状態変更を永続的に に配信します EventBridge。詳細については、「Amazon ユーザーガイド」の「Amazon EventBridgeとは」を参照してください。 EventBridge

トピック

- HealthImaging に送信されるイベント EventBridge
- HealthImaging イベント構造と例

HealthImaging に送信されるイベント EventBridge

次の表に、処理 EventBridge のために に送信されたすべての HealthImaging イベントを示します。

HealthImaging イベントタイプ	状態
データストアイベント	
データストアの作成	CREATING
データストアの作成に失敗しました	CREATE_FAILED
作成されたデータストア	ACTIVE
データストアの削除	DELETING
データストアが削除されました	DELETED
詳細については、AWS API リファレンスのdatastoreStatus」を参照してください。 HealthImaging	
ジョブイベントをインポートする	
インポートジョブが送信されました	SUBMITTED
進行中のインポートジョブ	IN_PROGRESS
インポートジョブが完了しました	COMPLETED
インポートジョブ失敗	FAILED
詳細については、AWS API リファレンスの jobStatus を参照してください。 HealthImaging	
画像セットイベント	
イメージセットが作成されました	CREATED
画像セットのコピー	COPYING
読み取り専用アクセスによる画像セットのコピー	COPYING_WITH_READ_ONLY_ACCESS

HealthImaging イベントタイプ	状態
画像セットのコピー	COPIED
画像セットのコピーに失敗しました	COPY_FAILED
画像セットの更新	UPDATING
画像セットの更新	UPDATED
画像セットの更新に失敗しました	UPDATE_FAILED
画像セットの削除	DELETING
削除された画像セット	DELETED

詳細については、AWS API リファレンスImageSetWorkflowStatusの「」を参照してください。 HealthImaging

HealthImaging イベント構造と例

HealthImaging イベントは、メタデータの詳細を含む JSON 構造を持つオブジェクトです。メタデータを入力として使用して、イベントを再作成するか、詳細情報を確認できます。関連するすべてのメタデータフィールドは、次のメニューのコード例の下の表に一覧表示されます。詳細については、「Amazon ユーザーガイド」の「イベント構造のリファレンス」を参照してください。 EventBridge

> **Note**
>
> HealthImaging イベント構造の source 属性は ですaws.medical-imaging。

データストアイベント

Data Store Creating

状態 - **CREATING**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
```

```
    "detail-type": "Data Store Creating",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "CREATING"
    }
}
```

Data Store Creation Failed

状態 - **CREATE_FAILED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Creation Failed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "CREATE_FAILED"
    }
}
```

Data Store Created

状態 - **ACTIVE**

```
{
    "version": "0",
```

```
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Created",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "ACTIVE"
    }
}
```

Data Store Deleting

状態 - **DELETING**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Deleting",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "DELETING"
    }
}
```

Data Store Deleted

状態 - **DELETED**

```
{
```

```
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Deleted",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "DELETED"
    }
}
```

データストアイベント - メタデータの説明

名前	型	説明
version	string	EventBridge イベントスキーマのバージョン。
id	string	すべてのイベントに対して生成されたバージョン 4 UUID。
detail-type	string	送信されるイベントのタイプ。
source	string	イベントを発生させたサービスを識別します。
account	string	データストア所有者の 12 桁の AWS アカウント ID。
time	string	イベントが発生した時刻。
region	string	データストアの AWS リージョンを識別します。

名前	型	説明
resources	配列 (文字列)	データストアの ARN を含む JSON 配列。
detail	オブジェクト	イベントに関する情報を含む JSON オブジェクト。
detail.imagingVersion	string	の HealthImagingイベント詳細スキーマへの変更を追跡するバージョン ID。
detail.datastoreId	string	ステータス変更イベントに関連付けられたデータストア ID。
detail.datastoreName	string	データストア名。
detail.datastoreStatus	string	現在のデータストアのステータス。

ジョブイベントをインポートする

Import Job Submitted

状態 - **SUBMITTED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Submitted",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
```

```
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "SUBMITTED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

Import Job In Progress

状態 - **IN_PROGRESS**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job In Progress",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "IN_PROGRESS",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

Import Job Completed

状態 - **COMPLETED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Completed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
```

```
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "COMPLETED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

Import Job Failed

状態 - **FAILED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Failed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "FAILED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

インポートジョブイベント - メタデータの説明

名前	型	説明
version	string	EventBridge イベントスキーマのバージョン。
id	string	すべてのイベントに対して生成されたバージョン 4 UUID。
detail-type	string	送信されるイベントのタイプ。
source	string	イベントを発生させたサービスを識別します。
account	string	データストア所有者の 12 桁の AWS アカウント ID。
time	string	イベントが発生した時刻。
region	string	データストアの AWS リージョンを識別します。
resources	配列 (文字列)	データストアの ARN を含む JSON 配列。
detail	オブジェクト	イベントに関する情報を含む JSON オブジェクト。
detail.imagingVersion	string	の HealthImagingイベント詳細スキーマへの変更を追跡するバージョン ID。
detail.datastoreId	string	ステータス変更イベントを生成したデータストア。
detail.jobId	string	ステータス変更イベントに関連付けられたインポートジョブ ID。

名前	型	説明
detail.jobName	string	インポートジョブ名。
detail.jobStatus	string	現在のジョブステータス。
detail.inputS3Uri	string	インポートする DICOM ファイルを含む S3 バケットの入力プレフィックスパス。
detail.outputS3Uri	string	DICOM インポートジョブの結果をアップロードする S3 バケットの出力プレフィックス。

画像セットイベント

Image Set Created

状態 - **CREATED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Created",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "CREATED"
   }
}
```

Image Set Copying

状態 - **COPYING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING"
   }
}
```

Image Set Copying With Read Only Access

状態 - **COPYING_WITH_READ_ONLY_ACCESS**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying With Read Only Access",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS"
   }
```

```
}
```

Image Set Copied

状態 - **COPIED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copied",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPIED"
   }
}
```

Image Set Copy Failed

状態 - **COPY_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copy Failed",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
```

```
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPY_FAILED"
   }
}
```

Image Set Updating

状態 - **UPDATING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updating",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "UPDATING"
   }
}
```

Image Set Updated

状態 - **UPDATED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updated",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
```

```
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATED"
   }
}
```

Image Set Update Failed

状態 - **UPDATE_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Update Failed",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATE_FAILED"
   }
}
```

Image Set Deleting

状態 - **DELETING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleting",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
```

```
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "DELETING"
   }
}
```

Image Set Deleted

状態 - **DELETED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleted",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "DELETED",
      "imageSetWorkflowStatus": "DELETED"
   }
}
```

画像セットイベント - メタデータの説明

名前	型	説明
`version`	string	EventBridge イベントスキーマのバージョン。

名前	型	説明
id	string	すべてのイベントに対して生成されたバージョン 4 UUID。
detail-type	string	送信されるイベントのタイプ。
source	string	イベントを発生させたサービスを識別します。
account	string	データストア所有者の 12 桁の AWS アカウント ID。
time	string	イベントが発生した時刻。
region	string	データストアの AWS リージョンを識別します。
resources	配列 (文字列)	画像セットの ARN を含む JSON 配列。
detail	オブジェクト	イベントに関する情報を含む JSON オブジェクト。
detail.imagingVersion	string	の HealthImagingイベント詳細スキーマへの変更を追跡するバージョン ID。
detail.datastoreId	string	ステータス変更イベントを生成したデータストア ID。
detail.imagesetId	string	ステータス変更イベントに関連付けられたイメージセット ID。
detail.imageSetState	string	現在の画像セットの状態。
detail.imageSetWorkflowStatus	string	現在の画像セットのワークフローステータス。

のセキュリティ AWS HealthImaging

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からメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ — AWS は、 で AWS サービスを実行するインフラストラクチャを保護する責任を担います AWS クラウド。また、 は、お客様が安全に使用できるサービス AWS も提供します。コンプライアンスAWS プログラム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者は定期的にセキュリティの有効性をテストおよび検証。に適用されるコンプライアンスプログラムの詳細については AWS HealthImaging、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、 を使用する際の責任共有モデルの適用方法を理解するのに役立ちます HealthImaging。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために HealthImaging を設定する方法を示します。また、 HealthImaging リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

トピック

- AWS でのデータ保護 HealthImaging
- AWS の Identity and Access Management HealthImaging
- AWS HealthImaging のコンプライアンス検証
- AWS HealthImaging のインフラストラクチャセキュリティ
- AWS CloudFormation による AWS HealthImaging リソースの作成
- AWS HealthImaging およびインターフェイス VPC エンドポイント (AWS PrivateLink)
- のクロスアカウントインポート AWS HealthImaging
- AWS HealthImaging のレジリエンス

AWS でのデータ保護 HealthImaging

責任 AWS 共有モデル、AWS でのデータ保護に適用されます HealthImaging。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。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のよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された記事「AWS 責任共有モデルおよび GDPR」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 Identity Center 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 は必須であり TLS 1.3 がお勧めです。
- 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。
- AWS 暗号化ソリューションと、 内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 などの高度なマネージド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 140-2 検証済みの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) 140-2」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は、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配置しないことを強くお勧めします。これは、コンソール、API、HealthImaging または SDK を使用して 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 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

トピック

- データ暗号化

- ネットワークトラフィックのプライバシー

データ暗号化

AWS を使用すると HealthImaging、クラウドに保管中のデータにセキュリティレイヤーを追加し、スケーラブルで効率的な暗号化機能を提供できます。具体的には次のとおりです。

- ほとんどの AWS サービスで利用可能な保管中のデータの暗号化機能
- AWS 暗号化キーを管理するか AWS Key Management Service、独自のキーを完全に制御するかを選択できる、 などの柔軟なキー管理オプション。
- AWS 所有の AWS KMS 暗号化キー
- Amazon SQS に対するサーバー側の暗号化 (SSE) を使用した、機密データを送信するための暗号化メッセージキュー

さらに、 は、暗号化とデータ保護を、お客様が開発または AWS 環境にデプロイする サービスと統合するための APIs AWS を提供します。

保管中の暗号化

HealthImaging は、デフォルトで暗号化を提供し、サービス所有の AWS KMS キーを使用して保管中の顧客の機密データを保護します。

転送中の暗号化

HealthImaging は TLS 1.2 を使用して、パブリックエンドポイントおよびバックエンドサービスを介して転送中のデータを暗号化します。

キー管理

AWS KMS キー (KMS キー) は、 のプライマリリソースです AWS Key Management Service。の外部で使用するデータキーを生成することもできます AWS KMS。

AWS 所有の KMS キー

HealthImaging は、デフォルトでこれらのキーを使用して、保管中の個人を特定できるデータやプライベートヘルス情報 (PHI) データなどの機密情報を自動的に暗号化します。 AWS 所有の KMS キーは アカウントに保存されません。これらは、複数の AWS アカウントで使用するために が AWS 所有および管理する KMS キーのコレクションの一部です。 AWS サービスは、 AWS が所有する KMS

キーを使用してデータを保護します。 AWS 所有の KMS キーを表示、管理、使用したり、その使用を監査したりすることはできません。ただし、データを暗号化するキーを保護するための作業やプログラムを操作したり変更したりする必要はありません。

AWS 所有の KMS キーを使用する場合、月額料金や使用料金は請求されず、アカウントの AWS KMS クォータにもカウントされません。詳細については、AWS Key Management Service デベロッパーガイド の「AWS 所有キー」を参照してください。

カスタマーマネージド KMS キー

HealthImaging は、ユーザーが作成、所有、管理する対称カスタマーマネージド KMS キーを使用して、既存の AWS 所有の暗号化に 2 番目の暗号化レイヤーを追加します。この暗号化レイヤーはユーザーが完全に制御できるため、次のようなタスクを実行できます。

- キーポリシー、IAM ポリシー、許可の確立と維持
- キー暗号化マテリアルのローテーション
- キーポリシーの有効化と無効化
- タグの追加
- キーエイリアスの作成
- 削除のためのキースケジューリング

を使用して CloudTrail 、 が AWS KMS ユーザーに代わって HealthImaging に送信するリクエストを追跡することもできます。 AWS KMS 追加料金が適用されます。詳細については、「AWS Key Management Service デベロッパーガイド」の「カスタマーマネージドキー」を参照してください。

カスタマーマネージドキーの作成

対称カスタマーマネージドキーは、 AWS Management Console または AWS KMS APIsを使用して作成できます。詳細については、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 KMS キーを作成する」を参照してください。

キーポリシーは、カスタマーマネージドキーへのアクセスを制御します。すべてのカスタマーマネージドキーには、キーポリシーが 1 つだけ必要です。このポリシーには、そのキーを使用できるユーザーとその使用方法を決定するステートメントが含まれています。カスタマーマネージドキーを作成する際に、キーポリシーを指定することができます。詳細については、AWS Key Management Service デベロッパーガイド の「カスタマーマネージドキーへのアクセスの管理」を参照してください。

HealthImaging リソースでカスタマーマネージドキーを使用するには、KMS:CreateGrant キーポリシーでオペレーションを許可する必要があります。これにより、指定された KMS キーへのアクセスを制御するカスタマーマネージドキーにグラントが追加され、ユーザーはグラントオペレーション HealthImaging に必要なアクセス権が付与されます。詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMSの許可」を参照してください。

HealthImaging リソースでカスタマーマネージド KMS キーを使用するには、キーポリシーで次の API オペレーションを許可する必要があります。

- kms:DescribeKey は、キーの検証に必要なカスタマーマネージドキーの詳細を提供します。これはすべてのオペレーションに必要です。
- kms:GenerateDataKey は、すべての書き込み操作で保存中の暗号化リソースにアクセスできるようにします。
- kms:Decrypt は暗号化されたリソースの読み取りまたは検索操作へのアクセスを提供します。
- kms:ReEncrypt* はリソースを再暗号化するためのアクセスを提供します。

以下は、ユーザーがそのキーによって HealthImaging 暗号化されたデータストアを作成して操作できるようにするポリシーステートメントの例です。

```
{
    "Sid": "Allow access to create data stores and perform CRUD and search in
 HealthImaging",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "medical-imaging.amazonaws.com"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:EncryptionContext:kms-arn": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f",
            "kms:EncryptionContext:aws:medical-imaging:datastoreId": "datastoreId"
        }
    }
```

```
}
```

カスタマーマネージド KMS キーの使用に必要な IAM アクセス許可

カスタマーマネージド KMS キーを使用して AWS KMS 暗号化を有効にしたデータストアを作成する場合、 HealthImaging データストアを作成するユーザーまたはロールのキーポリシーと IAM ポリシーの両方に必要なアクセス許可があります。

キーポリシーの詳細については、「AWS Key Management Service デベロッパーガイド」の「IAM ポリシーの有効化」を参照してください。

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`、`kms:GenerateDataKey`、、`kms:RetireGrant``kms:Decrypt`、および に対するアクセス許可に加えて`kms:ReEncrypt*`、AWS に必要なアクセス許可が必要です HealthImaging。

で 許可 HealthImaging を使用する方法 AWS KMS

HealthImaging には、カスタマーマネージド KMS キーを使用するための許可が必要です。カスタマーマネージド KMS キーで暗号化されたデータストアを作成すると、 は にCreateGrantリクエストを送信して、ユーザーに代わってグラント HealthImaging を作成します AWS KMS。のグラント AWS KMS は、顧客アカウントの KMS キー HealthImaging へのアクセスを許可するために使用されます。

がユーザーに代わって HealthImaging 作成する許可は、取り消したり廃止したりしないでください。アカウント内のキーを使用する AWS KMS アクセス HealthImaging 許可を付与する許可を取り消したり廃止したりする場合、このデータにアクセス HealthImaging したり、データストアにプッシュされた新しい画像リソースを暗号化したり、プル時に復号したりすることはできません。の許可を取り消すか廃止すると HealthImaging、変更はすぐに行われます。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、データストアを削除します。データストアが削除されると、 はユーザーに代わって許可を HealthImaging 廃止します。

HealthImaging の暗号化キーのモニタリング

CloudTrail カスタマー管理の KMS キーを使用するときに、 を使用して が AWS KMS ユーザーに代わって HealthImaging に送信するリクエストを追跡できます。ログの CloudTrail ログエントリは`userAgent`フィールド`medical-imaging.amazonaws.com`に表示され、 によって行われたリクエストを明確に区別します HealthImaging。

次の例はCreateGrant、カスタマーマネージドキーによって暗号化されたデータにアクセスDescribeKeyするために によって呼び出される AWS KMS オペレーションをモニタリング HealthImaging するための Decrypt、、、および GenerateDataKeyの CloudTrail イベントです。

以下では、 CreateGrantを使用して HealthImaging が顧客提供の KMS キーにアクセスできるようにする方法を示します。これにより、 はその KMS キー HealthImaging を使用して、保管中のすべての顧客データを暗号化できます。

ユーザーは、独自の grants を作成する必要はありません。 は、 にCreateGrantリクエストを送信することで、ユーザーに代わって grants HealthImaging を作成します AWS KMS。の許可 AWS KMS は、顧客アカウントの AWS KMS キー HealthImaging へのアクセスを許可するために使用されます。


```
{
    "Grants": [
        {
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "DescribeKey"
            ],
            "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
            "Name": "0a74e6ad2aa84b74a22fcd3efac1eaa8",
            "RetiringPrincipal": "AWS Internal",
            "GranteePrincipal": "AWS Internal",
            "GrantId":
 "0da169eb18ffd3da8c0eebc9e74b3839573eb87e1e0dce893bb544a34e8fbaaf",
            "IssuingAccount": "AWS Internal",
            "CreationDate": 1685050229.0,
            "Constraints": {
                "EncryptionContextSubset": {
                    "kms-arn": "arn:aws:kms:us-
west-2:824333766656:key/2fe3c119-792d-4b99-822f-b5841e1181d1"
                }
            }
        },
        {
            "Operations": [
                "GenerateDataKey",
```

```
                "CreateGrant",
                "RetireGrant",
                "DescribeKey"
            ],
            "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
            "Name": "2023-05-25T21:30:17",
            "RetiringPrincipal": "AWS Internal",
            "GranteePrincipal": "AWS Internal",
            "GrantId":
 "8229757abbb2019555ba64d200278cedac08e5a7147426536fcd1f4270040a31",
            "IssuingAccount": "AWS Internal",
            "CreationDate": 1685050217.0,
        }
    ]
}
```

以下の例は、GenerateDataKey を使用して、ユーザーがデータを暗号化するのに必要な許可を持っているか、データを保存する前に確認する方法を示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLEUSER",
        "arn": "arn:aws:sts::111122223333:assumed-role/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLEKEYID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLEROLE",
                "arn": "arn:aws:iam::111122223333:role/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2021-06-30T21:17:06Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "medical-imaging.amazonaws.com"
```

```
    },
    "eventTime": "2021-06-30T21:17:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "medical-imaging.amazonaws.com",
    "userAgent": "medical-imaging.amazonaws.com",
    "requestParameters": {
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
    },
    "responseElements": null,
    "requestID": "EXAMPLE_ID_01",
    "eventID": "EXAMPLE_ID_02",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

次の例は、 が Decryptオペレーションを HealthImaging 呼び出して、保存された暗号化されたデータキーを使用して暗号化されたデータにアクセスする方法を示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLEUSER",
        "arn": "arn:aws:sts::111122223333:assumed-role/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLEKEYID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLEROLE",
```

```
                "arn": "arn:aws:iam::111122223333:role/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2021-06-30T21:17:06Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "medical-imaging.amazonaws.com"
    },
    "eventTime": "2021-06-30T21:21:59Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "medical-imaging.amazonaws.com",
    "userAgent": "medical-imaging.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
    },
    "responseElements": null,
    "requestID": "EXAMPLE_ID_01",
    "eventID": "EXAMPLE_ID_02",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


次の例は、 が DescribeKeyオペレーション HealthImaging を使用して、 AWS KMS 顧客所有の AWS KMS キーが使用可能な状態であるかどうかを確認し、機能していない場合のユーザーのトラブルシューティングを支援する方法を示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLEUSER",
        "arn": "arn:aws:sts::111122223333:assumed-role/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLEKEYID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLEROLE",
                "arn": "arn:aws:iam::111122223333:role/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2021-07-01T18:36:14Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "medical-imaging.amazonaws.com"
    },
    "eventTime": "2021-07-01T18:36:36Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "medical-imaging.amazonaws.com",
    "userAgent": "medical-imaging.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
    },
    "responseElements": null,
    "requestID": "EXAMPLE_ID_01",
    "eventID": "EXAMPLE_ID_02",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
        }
```

```
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

詳細はこちら

以下のリソースは、保管中のデータの暗号化に関する詳細を説明しており、「AWS Key Management Service デベロッパーガイド」にあります。

- AWS KMS の概念
- のセキュリティのベストプラクティス AWS KMS

ネットワークトラフィックのプライバシー

トラフィックは HealthImaging 、 とオンプレミスアプリケーション間、および と Amazon S3 間で HealthImaging保護されます。 HealthImaging と 間のトラフィックは、デフォルトで HTTPS AWS Key Management Service を使用します。

- AWS HealthImaging は、米国東部 (バージニア北部) 、米国西部 (オレゴン) 、欧州 (アイルランド) 、アジアパシフィック (シドニー) の各リージョンで利用できるリージョンサービスです。
- HealthImaging と Amazon S3 バケット間のトラフィックの場合、Transport Layer Security (TLS) は HealthImaging と Amazon S3 の間で転送中のオブジェクトを暗号化し、それにアクセスする HealthImaging とカスタマーアプリケーションの間では、Amazon S3 バケット IAM ポリシー`aws:SecureTransport condition`の を使用して HTTPS (TLS) 経由の暗号化された接続のみを許可する必要があります。 HealthImaging 現在、 はパブリックエンドポイントを使用して Amazon S3 バケットのデータにアクセスしますが、データがパブリックインターネットを通過するわけではありません。 HealthImaging と Amazon S3 間のすべてのトラフィックは AWS ネットワーク経由でルーティングされ、TLS を使用して暗号化されます。

AWS の Identity and Access Management HealthImaging

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) し、誰に

HealthImaging 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 AWS のサービス 使用できる です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、 で行う作業によって異なります HealthImaging。

サービスユーザー – HealthImaging サービスを使用してジョブを実行する場合、管理者から必要な認証情報とアクセス許可が与えられます。さらに多くの HealthImaging 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。の機能にアクセスできない場合は、HealthImaging「」を参照してくださいAWS HealthImaging 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。

サービス管理者 – 社内の HealthImaging リソースを担当している場合は、通常、 へのフルアクセスがあります HealthImaging。サービスユーザーがどの HealthImaging 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で IAM を で使用する方法の詳細については、 HealthImaging「」を参照してくださいAWS と IAM HealthImaging の連携方法。

IAM 管理者 – IAM 管理者は、 へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります HealthImaging。IAM で使用できる HealthImaging アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS のアイデンティティベースのポリシーの例 HealthImaging。

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。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証（ にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 にサインインする方法 AWS アカウントAWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、アカウントのセキュリティを強化するために多要素認証 (MFA) を使用することをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「Multi-factor authentication」(多要素認証) および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」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、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。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。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、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。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリ、または ID ソースを通じて提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID がにアクセスすると AWS アカウント、ロールが引き受けられ、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center とは) 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、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションするを参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdminsという名前のグループを設定して、そのグループにIAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」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 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。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。
- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがありま

す。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、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ことができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、IAM ユーザーガイドのAmazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与するを参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、IAM ユーザーガイドの(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?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、 AWS ID 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ているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) AWS 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。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。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構造と内容の詳細については、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 AWS JSON 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んなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。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。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole`アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

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ポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。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 ポリシーの作成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、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたは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 ユーザーガイドのマネージドポリシーとインラインポリシーの比較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。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。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。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。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。プリンシパルには、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、または を含めることができます AWS のサービス。

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。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイドのアクセスコントロールリスト (ACL) の概要を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、IAM ユーザーガイドのIAM エンティティのアクセス許可の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 (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、AWS Organizations ユーザーガイドの「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。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権限を理解するのがさらに難しくなります。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。

AWS と IAM HealthImaging の連携方法

IAM を使用して へのアクセスを管理する前に HealthImaging、 で使用できる IAM 機能について学びます HealthImaging。

AWS で使用できる IAM の機能 HealthImaging

IAM 機能	HealthImaging サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	なし
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	なし
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	あり
プリンシパル権限	あり
サービスロール	あり
サービスリンクロール	なし

HealthImaging およびその他の AWS のサービスがほとんどの IAM 機能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「IAM AWS と連携する のサービス」を参照してください。

のアイデンティティベースのポリシー HealthImaging

アイデンティティベースポリシーをサポートする	あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。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 ポリシーの作成を参照してください。

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、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、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。

のアイデンティティベースのポリシーの例 HealthImaging

HealthImaging アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS のアイデンティティベースのポリシーの例 HealthImaging。

内のリソースベースのポリシー HealthImaging

リソースベースのポリシーのサポート	なし

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。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。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。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。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。プリンシパルには、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、または を含めることができます AWS のサービス。

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 にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) 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。IAM 管理者は、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権限を付与します。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をさらに付与する必要はありません。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。

のポリシーアクション HealthImaging

ポリシーアクションに対するサポート	あり

管理者は AWS JSON 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んなリソースにどんな条件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。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

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権限を付与するポリシーで使用されます。

HealthImaging アクションのリストを確認するには、「サービス認証リファレンス」の「AWS で定義されるアクション HealthImaging」を参照してください。

のポリシーアクションは、アクションの前に次のプレフィックス HealthImaging を使用します。

```
AWS
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
```

```
      "AWS:action1",
      "AWS:action2"
         ]
```

HealthImaging アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS のアイデンティティベースのポリシーの例 HealthImaging。

のポリシーリソース HealthImaging

ポリシーリソースに対するサポート	あり

管理者は AWS JSON 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どのような条件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。ステートメントには、`Resource` または `NotResource` 要素を含める必要があります。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。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限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": "*"
```

HealthImaging リソースタイプとその ARNs」の「AWS で定義されるリソースタイプ HealthImaging」を参照してください。 ARN を使用できるアクションとリソースについては、「AWS で定義されるアクション HealthImaging」を参照してください。

HealthImaging アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS のアイデンティティベースのポリシーの例 HealthImaging。

のポリシー条件キー HealthImaging

サービス固有のポリシー条件キーのサポート	あり

管理者は AWS JSON 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んなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。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することで、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。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、IAM ユーザーガイドのIAM ポリシーの要素: 変数およびタグを参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。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。

HealthImaging 条件キーのリストを確認するには、「サービス認証リファレンス」の「AWS の条件キー HealthImaging」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、「AWS で定義されるアクション HealthImaging」を参照してください。

HealthImaging アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS のアイデンティティベースのポリシーの例 HealthImaging。

ACLs HealthImaging

ACL のサポート	なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

での RBAC HealthImaging

RBAC のサポート	はい

IAM で使用される従来の認証モデルは、ロールベースのアクセスコントロール (RBAC) と呼ばれます。RBAC は、 の外部でロール AWS として知られる個人の職務機能に基づいてアクセス許可を定義します。詳細については、「IAM ユーザーガイド」の「ABAC と従来の RBAC モデルの比較」を参照してください。

での ABAC HealthImaging

ABAC (ポリシー内のタグ) のサポート	部分的

> **Warning**
>
> ABAC は SearchImageSets API アクション経由では適用されません。SearchImageSets アクションにアクセスできるユーザーであれば、データストア内の画像セットのすべてのメタデータにアクセスできます。

> **Note**
>
> 画像セットはデータストアの子リソースです。ABAC を使用するには、画像セットにデータストアと同じタグが必要です。詳細については、「AWS によるリソースのタグ付け HealthImaging」を参照してください。

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、aws: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。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、IAM ユーザーガイドのABAC とは?を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、IAM ユーザーガイドの属性に基づくアクセスコントロール (ABAC) を使用するを参照してください。

での一時的な認証情報の使用 HealthImaging

一時的な認証情報のサポート	あり

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドのAWS のサービス「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、IAM ユーザーガイドのロールへの切り替え (コンソール)を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して . AWS recommends にアクセスできます AWS。これは、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することを推奨しています。詳細については、IAM の一時的セキュリティ認証情報を参照してください。

のクロスサービスプリンシパル許可 HealthImaging

フォワードアクセスセッション (FAS) をサポート	あり

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。ポリシーによって、プリンシパルに許可が付与されます。一部のサービスを使用する際に、アクションを実行することで、別サービスの別アクションがトリガーされることがあります。この場合、両方のアクションを実行するための権限が必要です。アクションがポリシーで追加の依存アクションを必要とするかどうかを確認するには、「サービス認証リファレンス」の「AWS のアクション、リソース、および条件キー HealthImaging」を参照してください。

HealthImaging のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、 HealthImaging 機能が破損する可能性があります。が指示する場合以外 HealthImaging は、サービスロールを編集しないでください。

のサービスにリンクされたロール HealthImaging

サービスにリンクされたロールのサポート	なし

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、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ことができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスリンクロールの作成または管理の詳細については、IAM と提携するAWS のサービスを参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスリンクロールに関するドキュメントをサービスで表示するには、はい リンクを選択します。

AWS のアイデンティティベースのポリシーの例 HealthImaging

デフォルトでは、ユーザーとロールにはリソースを作成または変更 HealthImagingするアクセス許可はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。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。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの 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 ポリシーの作成を参照してください。

各リソースタイプの ARNs「サービス認証リファレンス」の「Awesome AWS 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。

トピック

- ポリシーのベストプラクティス
- HealthImaging コンソールを使用する
- 自分の権限の表示をユーザーに許可する

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で誰かが HealthImaging 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を実行すると、 AWS アカウントに料金が発生する可能性があります。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を付与するには、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して、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。例えば、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、 などの特定の を介してサービスアクションが使用される場合に 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 AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素:条件) を参照してください。

- IAM Access Analyzer 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 Access Analyzer 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 ポリシーの検証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、セキュリティを強化するために MFA を有効にします。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 保護 API アクセスの設定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。

HealthImaging コンソールを使用する

AWS HealthImaging コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の HealthImaging リソースの詳細を一覧表示および表示できます AWS アカウント。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。

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。

ユーザーとロールが HealthImaging 引き続き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 HealthImaging *ConsoleAccess*または *ReadOnly* AWS 管理ポリシーもアタッチします。詳細については、IAM ユーザーガイドのユーザーへの許可の追加を参照してください。

自分の権限の表示をユーザーに許可する

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": "*"
        }
    ]
}
```

AWS HealthImaging の AWS マネージドポリシー

AWSマネージドポリシーは、AWSが作成および管理するスタンドアロンポリシーです。AWSマネージドポリシーは、多くの一般的なユースケースでアクセス許可を提供できるように設計されているため、ユーザー、グループ、ロールへのアクセス許可の割り当てを開始できます。

AWSマネージドポリシーは、ご利用の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ご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。ユースケース別にカスタマーマネージドポリシーを定義することで、アクセス許可を絞り込むことをお勧めします。

AWS マネージドポリシーで定義したアクセス権限は変更できません。AWSがAWSマネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しいAWS のサービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWSがAWSマネージドポリシーを更新する可能性が最も高くなります。

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。

トピック

- AWS マネージドポリシー: AWSHealthImagingFullAccess
- AWS マネージドポリシー: AWSHealthImagingReadOnlyAccess
- HealthImaging のAWS マネージドポリシーに対する更新

AWS マネージドポリシー: AWSHealthImagingFullAccess

`AWSHealthImagingFullAccess` ポリシーは IAM ID に添付できます。

このポリシーは、HealthImaging のすべてのアクションに管理者権限を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "medical-imaging:*"
            ],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "medical-imaging.amazonaws.com"
                }
            }
        }
    ]
}
```

AWS マネージドポリシー: AWSHealthImagingReadOnlyAccess

`AWSHealthImagingReadOnlyAccess` ポリシーは IAM ID に添付できます。

このポリシーは、特定の AWS HealthImaging アクションに対し読み取り専用アクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "medical-imaging:GetDICOMImportJob",
            "medical-imaging:GetDatastore",
            "medical-imaging:GetImageFrame",
            "medical-imaging:GetImageSet",
            "medical-imaging:GetImageSetMetadata",
            "medical-imaging:ListDICOMImportJobs",
            "medical-imaging:ListDatastores",
            "medical-imaging:ListImageSetVersions",
            "medical-imaging:ListTagsForResource",
            "medical-imaging:SearchImageSets"
        ],
        "Resource": "*"
    }]
}
```

HealthImaging のAWS マネージドポリシーに対する更新

このサービスがこれらの変更の追跡を開始してからの、HealthImaging の AWS マネージドポリシーに対する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については、リリース ページの RSS フィードを購読してください。

変更	説明	日付
HealthImaging が変更の追跡を開始	HealthImaging が AWS マネージドポリシーの変更の追跡を開始しました。	2023 年 7 月 19 日

AWS HealthImaging 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、 および IAM の使用時に発生する可能性がある一般的な問題の診断 HealthImaging と修正に役立ちます。

トピック

- でアクションを実行する権限がない HealthImaging
- iam を実行する権限がありません。PassRole
- 自分の 以外のユーザーに自分の HealthImaging リソース AWS アカウント へのアクセスを許可したい

でアクションを実行する権限がない HealthImaging

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、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AWS:*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:
 AWS:GetWidget on resource: my-example-widget
```

この場合、AWS:*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 を実行する権限がありません。PassRole

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 にロールを渡すことができるようにする必要があります HealthImaging。

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、そのサービスに既存のロールを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権限が必要です。

次の例のエラーは、 という IAM marymajor ユーザーがコンソールを使用して でアクションを実行しようする場合に発生します HealthImaging。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権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
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。サインイン認証情報を提供した担当者が管理者です。

自分の 以外のユーザーに自分の HealthImaging 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:

- がこれらの機能 HealthImaging をサポートしているかどうかを確認するには、「」を参照してくださいAWS と IAM HealthImaging の連携方法。

- 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でのロールとリソースベースのポリシーの使用の違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」を参照してください。

AWS HealthImaging のコンプライアンス検証

サードパーティーの監査者は、さまざまな AWS コンプライアンスプログラムの一環として AWS HealthImaging のセキュリティとコンプライアンスを評価します。HealthImagingの場合、これには HIPAA が含まれます。

特定のコンプライアンスプログラムの対象となる AWS サービスのリストについては、「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。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。

AWS Artifact 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。詳細については、「AWS Artifact にレポートをダウンロードする」を参照してください。

AWS HealthImaging を使用する際のユーザーのコンプライアンス責任は、ユーザーのデータの機密性や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。AWS では、コンプライアンスに役立つ以下のリソースを提供しています。

- AWSパートナーソリューション – セキュリティおよびコンプライアンスの自動化リファレンスデプロイガイドは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境を AWS にデプロイするための手順を説明します。
- Architecting for HIPAA Security and Compliance Whitepaper (HIPAA のセキュリティとコンプライアンスのためのアーキテクチャの設計に関するホワイトペーパー) - このホワイトペーパーは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWS における GxP システム — このホワイトペーパーでは、GxP 関連のコンプライアンスとセキュリティに対する AWS の取り組み方についての情報を提供し、GxP の観点における AWS サービスの利用に関するガイダンスを提供します。

- AWS コンプライアンスのリソース - このワークブックとガイドのコレクションは、お客様の業界や所在地に適用される場合があります。
- ルールを使用したリソースの評価 – AWS Config がリソース設定が社内のプラクティス、業界のガイドライン、規制にどの程度適合しているかを評価します。
- AWS Security Hub - この AWS サービスでは、AWS 内のセキュリティ状態を包括的に表示しており、セキュリティ業界の標準およびベストプラクティスへの準拠を確認するのに役立ちます。

AWS HealthImaging のインフラストラクチャセキュリティ

マネージド型サービスである AWS HealthImaging は、ホワイトペーパー「Amazon Web Services: のセキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティ対策により保護されています。

AWS が発行している APIコールを使用して、ネットワーク経由で HealthImaging にアクセスします。クライアントは、Transport Layer Security (TLS) 1.3 以降をサポートする必要があります。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

AWS CloudFormation による AWS HealthImaging リソースの作成

AWS HealthImaging は、リソースとインフラストラクチャの作成と管理の所要時間を短縮できるように AWS リソースをモデル化して設定するためのサービスである AWS CloudFormation と統合されています。必要なすべての AWS リソースを記述したテンプレートを作成すれば、AWS CloudFormation がこれらのリソースのプロビジョニングや設定を処理します。

AWS CloudFormation を使用すると、テンプレートを再利用して HealthImaging リソースをいつでも繰り返しセットアップできます。リソースを一度記述するだけで、同じリソースを複数の AWS アカウント とリージョンで何度でもプロビジョニングできます。

HealthImaging と AWS CloudFormation テンプレート

HealthImaging および関連サービスのリソースをプロビジョニングして設定するには、AWS CloudFormation テンプレートについて理解しておく必要があります。テンプレートは、JSON またはYAMLでフォーマットされたテキストファイルです。これらのテンプレートには、AWS CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSON や YAML に不慣れな方は、AWS CloudFormation Designer を使えば、AWS CloudFormation テンプレートを使いこなすことができます。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation Designer とは」を参照してください。

AWS HealthImaging は、AWS CloudFormation を使用したデータストアの作成をサポートしています。HealthImaging データストアのプロビジョニング用の JSON テンプレートと YAML テンプレートの例を含む詳細情報については、「AWS CloudFormationユーザーガイド」の「AWS HealthImagingリソースタイプのリファレンス」を参照してください。

AWS CloudFormation の詳細はこちら

AWS CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation API リファレンス
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

AWS HealthImaging およびインターフェイス 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 を作成 AWS HealthImaging することで、VPC と の間にプライベート接続を確立できます。インターフェイスエンドポイントは、インターネットゲートウェイAWS PrivateLink、NAT デバイス、VPN 接続、または AWS Direct Connect 接続なしで HealthImaging APIs にプライベートにアクセスできるテクノロジーである を利用しています。VPC 内のインスタンスは、パブリック IP アドレスがなくても HealthImaging APIs。VPC と 間のトラフィック HealthImaging は Amazon ネットワークを離れません。

各インターフェースエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink) 」を参照してください。

トピック

- HealthImaging VPC エンドポイントに関する考慮事項
- のインターフェイス VPC エンドポイントの作成 HealthImaging
- の VPC エンドポイントポリシーの作成 HealthImaging

HealthImaging VPC エンドポイントに関する考慮事項

のインターフェイス VPC エンドポイントを設定する前に HealthImaging、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントのプロパティと制限」を確認してください。

HealthImaging は、VPC からのすべての AWS HealthImaging アクションの呼び出しをサポートします。

のインターフェイス VPC エンドポイントの作成 HealthImaging

Amazon VPC コンソールまたは AWS Command Line Interface () を使用して、 HealthImaging サービスの VPC エンドポイントを作成できますAWS CLI。詳細については、 Amazon VPC ユーザーガイド のインターフェイスエンドポイントの作成を参照してください。

次のサービス名 HealthImaging を使用して 用の VPC エンドポイントを作成します。

- com.amazonaws.*region*.medical-imaging
- com.amazonaws.*region*runtime-medical-imaging
- com.amazonaws.*region*dicom-medical-imaging

> **Note**
>
> を使用するには、プライベート DNS を有効にする必要があります PrivateLink。

リージョンのデフォルトの DNS 名 HealthImaging を使用して、 に対して API リクエストを行うことができます。例えば、 ですmedical-imaging.us-east-1.amazonaws.com。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

の VPC エンドポイントポリシーの作成 HealthImaging

へのアクセスを制御するエンドポイントポリシーを VPC エンドポイントにアタッチできます HealthImaging。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル
- 実行可能なアクション
- アクションを実行できるリソース

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

例: HealthImaging アクションの VPC エンドポイントポリシー

のエンドポイントポリシーの例を次に示します HealthImaging。このポリシーは、エンドポイントにアタッチされると、すべてのリソースのすべてのプリンシパルに対して HealthImaging アクションへのアクセスを許可します。

API

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "medical-imaging:*"
         ],
         "Resource":"*"
      }
   ]
}
```

CLI

```
aws ec2 modify-vpc-endpoint \
    --vpc-endpoint-id vpce-id
    --region us-west-2 \
    --private-dns-enabled \
    --policy-document \
```

```
    "{\"Statement\":[{\"Principal\":\"*\",\"Effect\":\"Allow\",\"Action\":
[\"medical-imaging:*\"],\"Resource\":\"*\"}]}"
```

のクロスアカウントインポート AWS HealthImaging

クロスアカウント/クロスリージョンインポートを使用すると、サポートされている他のリージョンにある Amazon S3 バケットから HealthImaging データストアにデータをインポートできます。AWS アカウント間、他の AWS Organizations が所有するアカウント間、および の「オープンデータレジストリ」にある「イメージングデータコモンズ (IDC)」などのオープンデータソースからデータをインポートできます。 AWS

HealthImaging クロスアカウント/クロスリージョンインポートのユースケースには以下が含まれます。

- 顧客アカウントから DICOM データをインポートする医療画像 SaaS 製品
- 多くの Amazon S3 入力バケットから 1 つの HealthImaging データストアを入力する大規模な組織
- 研究者が多施設臨床試験間でデータを安全に共有する

クロスアカウントインポートを使用するには

1. Amazon S3 入力 (ソース) バケットの所有者は、 HealthImaging データストアの所有者`s3:ListBucket`と`s3:GetObject`アクセス許可を付与する必要があります。
2. HealthImaging データストア所有者は、Amazon S3 バケットを IAM に追加する必要がありますImportJobDataAccessRole。インポート用の IAM ロールの作成 を参照してください。
3. HealthImaging データストア所有者は、インポートジョブを開始するときに`inputOwnerAccountId` Amazon S3 入力バケットの を指定する必要があります。

> **Note**
>
> を指定することで`inputOwnerAccountId`、データストア所有者は、業界標準への準拠を維持し、潜在的なセキュリティリスクを軽減するために、入力 Amazon S3 バケットが指定されたアカウントに属していることを検証します。

次のstartDICOMImportJobコード例には、 インポートジョブの開始セクションのすべての AWS CLI および SDK コード例に適用できるオプションの inputOwnerAccountIdパラメータが含まれています。

Java

```
public static String startDicomImportJob(MedicalImagingClient
 medicalImagingClient,
        String jobName,
        String datastoreId,
        String dataAccessRoleArn,
        String inputS3Uri,
        String outputS3Uri,
        String inputOwnerAccountId) {

    try {
        StartDicomImportJobRequest startDicomImportJobRequest =
 StartDicomImportJobRequest.builder()
                .jobName(jobName)
                .datastoreId(datastoreId)
                .dataAccessRoleArn(dataAccessRoleArn)
                .inputS3Uri(inputS3Uri)
                .outputS3Uri(outputS3Uri)
                .inputOwnerAccountId(inputOwnerAccountId)
                .build();
        StartDicomImportJobResponse response =
 medicalImagingClient.startDICOMImportJob(startDicomImportJobRequest);
        return response.jobId();
    } catch (MedicalImagingException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return "";
}
```

AWS HealthImaging のレジリエンス

AWS グローバルインフラストラクチャは AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。AWS リージョン には、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている物理的に独立・隔離された複数のアベイラビリティーゾーンがあります。アベイラビリティーゾーンを使用すると、中断することなくゾーン間で自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用できます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョン とアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

AWS HealthImaging では、AWS グローバルインフラストラクチャに加えて、データの耐障害性とバックアップのニーズに対応できるように複数の機能を提供しています。

AWS HealthImaging リファレンスマテリアル

AWS では、次のリファレンス資料を使用できます HealthImaging。

> **Note**
>
> すべての HealthImaging アクションとデータ型は、別のリファレンスにあります。詳細については、「AWS HealthImaging API リファレンス」を参照してください。

トピック

AWS の DICOM サポート HealthImaging

AWS HealthImaging は、特定の DICOM 要素と転送構文をサポートしています。 HealthImaging メタデータキーはそれらに基づいているため、サポートされている患者、治験、シリーズレベルの DICOM データ要素を理解してください。インポートを開始する前に、医療画像データが HealthImagingでサポートされている転送構文と DICOM 要素の制約に準拠していることを確認してください。

> **Note**
>
> AWS HealthImaging は現在、バイナリセグメンテーションイメージまたはアイコンイメージシーケンスのピクセルデータをサポートしていません。

トピック

- メタデータの正規化
- サポートされる転送構文
- DICOM 要素の制約
- DICOM メタデータの制約

サポートされている SOP クラス

AWS を使用すると HealthImaging、廃止された やプライベート など、任意の SOP クラス UID でエンコードされた DICOM P10 サービスオブジェクトペア (SOP) インスタンスをインポートできます。すべてのプライベート属性も保持されます。

メタデータの正規化

DICOM P10 データを AWS にインポートすると HealthImaging、メタデータと画像フレーム (ピクセルデータ) で構成される画像セットに変換されます。 ???変換プロセス中に、 HealthImaging 特定のバージョンの DICOM 標準に基づいてメタデータキーが生成されます。 HealthImaging は現在、DICOM PS3.6 2022b データディクショナリ に基づいてメタデータキーを生成し、サポートしています。

AWS は、患者、治験、シリーズレベルで次の DICOM データ要素 HealthImaging をサポートしています。

患者レベルの要素

> **Note**
>
> 各患者レベル要素の詳細については、DICOM データ要素のレジストリを参照してください。

AWS では、以下の患者レベルの要素 HealthImaging がサポートされています。

```
Patient Module Elements

  (0010,0010) - Patient's Name
  (0010,0020) - Patient ID

Issuer of Patient ID Macro Elements
```

```
  (0010,0021) - Issuer of Patient ID
  (0010,0024) - Issuer of Patient ID Qualifiers Sequence
  (0010,0022) - Type of Patient ID
  (0010,0030) - Patient's Birth Date
  (0010,0033) - Patient's Birth Date in Alternative Calendar
  (0010,0034) - Patient's Death Date in Alternative Calendar
  (0010,0035) - Patient's Alternative Calendar Attribute
  (0010,0040) - Patient's Sex
  (0010,1100) - Referenced Patient Photo Sequence
  (0010,0200) - Quality Control Subject
  (0008,1120) - Referenced Patient Sequence
  (0010,0032) - Patient's Birth Time
  (0010,1002) - Other Patient IDs Sequence
  (0010,1001) - Other Patient Names
  (0010,2160) - Ethnic Group
  (0010,4000) - Patient Comments
  (0010,2201) - Patient Species Description
  (0010,2202) - Patient Species Code Sequence Attribute
  (0010,2292) - Patient Breed Description
  (0010,2293) - Patient Breed Code Sequence
  (0010,2294) - Breed Registration Sequence Attribute
  (0010,0212) - Strain Description
  (0010,0213) - Strain Nomenclature Attribute
  (0010,0219) - Strain Code Sequence
  (0010,0218) - Strain Additional Information Attribute
  (0010,0216) - Strain Stock Sequence
  (0010,0221) - Genetic Modifications Sequence Attribute
  (0010,2297) - Responsible Person
  (0010,2298) - Responsible Person Role Attribute
  (0010,2299) - Responsible Organization
  (0012,0062) - Patient Identity Removed
  (0012,0063) - De-identification Method
  (0012,0064) - De-identification Method Code Sequence

Patient Group Macro Elements

  (0010,0026) - Source Patient Group Identification Sequence
  (0010,0027) - Group of Patients Identification Sequence

Clinical Trial Subject Module

  (0012,0010) - Clinical Trial Sponsor Name
  (0012,0020) - Clinical Trial Protocol ID
  (0012,0021) - Clinical Trial Protocol Name Attribute
```

```
  (0012,0030) - Clinical Trial Site ID
  (0012,0031) - Clinical Trial Site Name
  (0012,0040) - Clinical Trial Subject ID
  (0012,0042) - Clinical Trial Subject Reading ID
  (0012,0081) - Clinical Trial Protocol Ethics Committee Name
  (0012,0082) - Clinical Trial Protocol Ethics Committee Approval Number
```

治験レベルの要素

> **Note**
>
> 各治験レベル要素の詳細については、DICOM データ要素のレジストリを参照してください。

AWS では、以下の治験レベルの要素 HealthImaging がサポートされています。

```
General Study Module

  (0020,000D) - Study Instance UID
  (0008,0020) - Study Date
  (0008,0030) - Study Time
  (0008,0090) - Referring Physician's Name
  (0008,0096) - Referring Physician Identification Sequence
  (0008,009C) - Consulting Physician's Name
  (0008,009D) - Consulting Physician Identification Sequence
  (0020,0010) - Study ID
  (0008,0050) - Accession Number
  (0008,0051) - Issuer of Accession Number Sequence
  (0008,1030) - Study Description
  (0008,1048) - Physician(s) of Record
  (0008,1049) - Physician(s) of Record Identification Sequence
  (0008,1060) - Name of Physician(s) Reading Study
  (0008,1062) - Physician(s) Reading Study Identification Sequence
  (0032,1033) - Requesting Service
  (0032,1034) - Requesting Service Code Sequence
  (0008,1110) - Referenced Study Sequence
  (0008,1032) - Procedure Code Sequence
  (0040,1012) - Reason For Performed Procedure Code Sequence

Patient Study Module

  (0008,1080) - Admitting Diagnoses Description
```

```
  (0008,1084) - Admitting Diagnoses Code Sequence
  (0010,1010) - Patient's Age
  (0010,1020) - Patient's Size
  (0010,1030) - Patient's Weight
  (0010,1022) - Patient's Body Mass Index
  (0010,1023) - Measured AP Dimension
  (0010,1024) - Measured Lateral Dimension
  (0010,1021) - Patient's Size Code Sequence
  (0010,2000) - Medical Alerts
  (0010,2110) - Allergies
  (0010,21A0) - Smoking Status
  (0010,21C0) - Pregnancy Status
  (0010,21D0) - Last Menstrual Date
  (0038,0500) - Patient State
  (0010,2180) - Occupation
  (0010,21B0) - Additional Patient History
  (0038,0010) - Admission ID
  (0038,0014) - Issuer of Admission ID Sequence
  (0032,1066) - Reason for Visit
  (0032,1067) - Reason for Visit Code Sequence
  (0038,0060) - Service Episode ID
  (0038,0064) - Issuer of Service Episode ID Sequence
  (0038,0062) - Service Episode Description
  (0010,2203) - Patient's Sex Neutered

Clinical Trial Study Module

  (0012,0050) - Clinical Trial Time Point ID
  (0012,0051) - Clinical Trial Time Point Description
  (0012,0052) - Longitudinal Temporal Offset from Event
  (0012,0053) - Longitudinal Temporal Event Type
  (0012,0083) - Consent for Clinical Trial Use Sequence
```

シリーズレベルの要素

> **Note**
>
> 各シリーズレベル要素の詳細については、DICOM データ要素のレジストリを参照してください。

AWS では、以下のシリーズレベルの要素 HealthImaging がサポートされています。

General Series Module

```
  (0008,0060) - Modality
  (0020,000E) - Series Instance UID
  (0020,0011) - Series Number
  (0020,0060) - Laterality
  (0008,0021) - Series Date
  (0008,0031) - Series Time
  (0008,1050) - Performing Physician's Name
  (0008,1052) - Performing Physician Identification Sequence
  (0018,1030) - Protocol Name
  (0008,103E) - Series Description
  (0008,103F) - Series Description Code Sequence
  (0008,1070) - Operators' Name
  (0008,1072) - Operator Identification Sequence
  (0008,1111) - Referenced Performed Procedure Step Sequence
  (0008,1250) - Related Series Sequence
  (0018,0015) - Body Part Examined
  (0018,5100) - Patient Position
  (0028,0108) - Smallest Pixel Value in Series
  (0028,0109) - Largest Pixel Value in Series
  (0040,0275) - Request Attributes Sequence
  (0010,2210) - Anatomical Orientation Type
  (300A,0700) - Treatment Session UID
```

Clinical Trial Series Module

```
  (0012,0060) - Clinical Trial Coordinating Center Name
  (0012,0071) - Clinical Trial Series ID
  (0012,0072) - Clinical Trial Series Description
```

General Equipment Module

```
  (0008,0070) - Manufacturer
  (0008,0080) - Institution Name
  (0008,0081) - Institution Address
  (0008,1010) - Station Name
  (0008,1040) - Institutional Department Name
  (0008,1041) - Institutional Department Type Code Sequence
  (0008,1090) - Manufacturer's Model Name
  (0018,100B) - Manufacturer's Device Class UID
  (0018,1000) - Device Serial Number
  (0018,1020) - Software Versions
```

```
  (0018,1008) - Gantry ID
  (0018,100A) - UDI Sequence
  (0018,1002) - Device UID
  (0018,1050) - Spatial Resolution
  (0018,1200) - Date of Last Calibration
  (0018,1201) - Time of Last Calibration
  (0028,0120) - Pixel Padding Value

Frame of Reference Module

  (0020,0052) - Frame of Reference UID
  (0020,1040) - Position Reference Indicator
```

サポートされる転送構文

AWS は、次の表にある転送構文でエンコードされた DICOM P10 SOP インスタンス HealthImaging をインポートします。は、SOP インスタンスのストレージに加えて、次の転送構文でエンコードされた SOP インスタンスの画像 HealthImaging フレーム (ピクセルデータ) を HTJ2K にトランスコードします。

転送構文 UID	転送構文名
1.2.840.10008.1.2	インプリシット VR エンディアン: DICOM のデフォルト転送構文
1.2.840.10008.1.2.1	エクスプリシット VR リトルエンディアン
1.2.840.10008.1.2.1.99	デフレートされたエクスプリシット VR リトルエンディアン
1.2.840.10008.1.2.2	エクスプリシット VR ビッグエンディアン
1.2.840.10008.1.2.4.50	JPEG ベースライン (プロセス 1): 不可逆JPEG 8 ビット画像圧縮のデフォルト転送構文
1.2.840.10008.1.2.4.51	JPEG ベースライン (プロセス 2 と 4): 不可逆 JPEG 12 ビット画像圧縮のデフォルト転送構文 (プロセス 4 のみ)
1.2.840.10008.1.2.4.57	JPEG 可逆非階層型 (プロセス 14)

転送構文 UID	転送構文名
1.2.840.10008.1.2.4.70	JPEG 可逆性、非階層型、一次予測 (プロセス 14 [選択値 1]): 可逆 JPEG 画像圧縮のデフォルト転送構文
1.2.840.10008.1.2.4.80	JPEG-LS 可逆画像圧縮
1.2.840.10008.1.2.4.81	JPEG-LS 不可逆 (ほぼ可逆性の) 画像圧縮
1.2.840.10008.1.2.4.90	JPEG 2000 画像圧縮 (可逆のみ)
1.2.840.10008.1.2.4.91	JPEG 2000 イメージ圧縮
1.2.840.10008.1.2.4.201	高スループット JPEG 2000 イメージ圧縮 (ロスレスのみ)
1.2.840.10008.1.2.4.202	RPCL オプションによる高スループット JPEG 2000 イメージ圧縮 (ロスレスのみ)
1.2.840.10008.1.2.4.203	高スループット JPEG 2000 イメージ圧縮
1.2.840.10008.1.2.5	RLE 可逆性

DICOM 要素の制約

医療画像データを AWS にインポートする場合 HealthImaging、最大長の制約が次の DICOM 要素に適用されます。インポートを正常に行うには、データが最大制限長を超えないようにしてください。

インポート中の DICOM 要素の制約

HealthImaging キーワード	DICOM キーワード	DICOM キー	長さ制限
DICOMPatientName	PatientName	(0010,0010)	最小: 0、最大: 256
DICOMPatientId	PatientID	(0010,0020)	最小: 0、最大: 256
DICOMPatientBirthDate	患者BirthDate	(0010,0030)	最小: 0、最大: 18

HealthImaging キーワード	DICOM キーワード	DICOM キー	長さ制限
DICOMPatientSex	PatientSex	(0010,0040)	最小: 0、最大: 16
DICOM StudyInstanceUID	StudyInstanceUID	(0020,000D)	最小: 0、最大: 64
DICOMStudyId	StudyID	(0020,0010)	最小: 0、最大: 16
DICOMStudyDescription	StudyDescription	(0008,1030)	最小: 0、最大: 64
DICOMNumberOfStudy RelatedSeries	NumberOfStudyRelatedSeries	(0020,1206)	最小: 0、最大: 10000
DICOMNumberOfStudy RelatedInstances	NumberOfStudyRelated インスタンス	(0020,1208)	最小: 0、最大: 10000
DICOMAccessionNumber	AccessionNumber	(0008,0050)	最小: 0、最大: 256
DICOMStudyDate	StudyDate	(0008,0020)	最小: 0、最大: 18
DICOMStudyTime	StudyTime	(0008,0030)	最小: 0、最大: 28

DICOM メタデータの制約

`UpdateImageSetMetadata` を使用して HealthImaging メタデータ属性を更新する場合、次の DICOM 制約が適用されます。

- 更新制約が `updatableAttributes`と の両方に適用されない限り、患者/治験/シリーズ/インスタンスレベルの属性のプライベート属性を更新または削除することはできません `removableAttributes`

- 次の AWS HealthImaging 生成属性を更新できません: `SchemaVersion`、`DatastoreIDImageSetID`、`PixelData`、、`Checksum`、`Width`、`Height`、`MinPi` `FrameSizeInBytes`
- 次の DICOM 属性は更新できません: `Tag.PixelData`、`Tag.StudyInstanceUID`、`Tag.SeriesInstanceUID`、`Tag.SOPInstanceUID`、
- VR タイプ SQ の属性 (ネスト属性) は更新できません
- 複数値を持つ属性は更新できません
- 属性 VR タイプと互換性のない値の属性は更新できません
- DICOM 標準で有効属性とみなされない属性は更新できません
- モジュール間で属性を更新することはできません 例えば、顧客ペイロードリクエストの治験レベルで患者レベルの属性が指定されている場合、そのリクエストは無効になる可能性があります。
- 関連する属性モジュールが`ImageSetMetadata`にない場合、属性を更新できません。例えば、`seriesInstanceUID` のシリーズが既存の画像セットにない場合、`seriesInstanceUID` の属性を更新できません。

AWS HealthImaging ピクセルデータ検証

インポート中、 は、すべてのイメージの可逆エンコードとデコード状態をチェックすることで、組み込みのピクセルデータ検証 HealthImaging を提供します。この機能により、HTJ2K デコードライブラリを使用してデコードされたイメージが、 にインポートされた元の DICOM P10 イメージと常に一致します HealthImaging。

- イメージオンボーディングプロセスは、インポートジョブがインポート前に DICOM P10 イメージの元のピクセル品質状態をキャプチャしたときに開始されます。CRC32 アルゴリズムを使用して、画像ごとに一意の変更不可能な画像フレーム解像度チェックサム (IFRC) が生成されます。IFRC は、各イメージのピクセルデータの解像度レベルごとに計算されます。IFRC チェックサム値はメタデータドキュメント (`job-output-manifest.json`) に表示され、ベースからフル解像度までリストでソートされます。
- イメージを HealthImaging データストアにインポートして画像セットに変換すると、HTJ2K-encoded画像フレームがすぐにデコードされ、新しい IFRCs が計算されます。 HealthImaging 次に、元の画像のフル解像度 IFRCs とインポートされた画像フレームの新しい IFRCs を比較して、精度を検証します。
- 対応する画像ごとの説明的なエラー条件がインポートジョブ出力ログ (`job-output-manifest.json`) にキャプチャされ、確認および検証できます。

ピクセルデータを検証するには

1. 医療画像データをインポートしたら、インポートジョブの出力ログである job-output-manifest.json に記録された、各画像セットの成功 (またはエラー状態) を確認できます。詳細については、「インポートジョブを理解する」を参照してください。
2. 画像セットは、メタデータと画像フレーム (ピクセルデータ) で構成されます。画像セットメタデータには、関連する画像フレームに関する情報が含まれています。GetImageSetMetadata アクションを使用して、画像セットのメタデータを取得します。詳細については、「画像セットメタデータの取得」を参照してください。
3. PixelDataChecksumFromBaseToFullResolution には解像度レベルごとの IFRC (チェックサム) が含まれています。以下は、インポートジョブプロセスの一部として生成され、 に記録された IFRC のメタデータ出力の例ですjob-output-manifest.json。

```
"ImageFrames": [{
  "ID": "678906789067890123451234512345 12",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 128,
      "Height": 128,
      "Checksum": 2928338830
    },
    {
      "Width": 256,
      "Height": 256,
      "Checksum": 1362274918
    },
    {
      "Width": 512,
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
```

4. ピクセルデータを検証するには、 のピクセルデータ検証手順にアクセスし GitHub 、README.md ファイルの指示に従って、 HTJ2K デコーディングライブラリ が使用するさまざまな による可逆画像処理を個別に検証します HealthImaging。データが解像度レベルごとに徐々にロードされるので、raw 入力データの IFRC を最後に計算し、同じ解像度の HealthImaging メタデータで提供される IFRC 値と比較してピクセルデータを検証できます。

AWS 用の HTJ2K デコードライブラリ HealthImaging

のインポート中、AWS はすべてのイメージフレーム (ピクセルデータ) を高スループット JPEG 2000 (HTJ2K) 可逆形式で HealthImaging エンコードし、一貫した高速なイメージ表示と HTJ2K の高度な機能へのユニバーサルアクセスを提供します。イメージフレームはインポート時に HTJ2K でエンコードされるため、イメージビューワーで表示する前にデコードする必要があります。

> **Note**
>
> HTJ2K は、JPEG2000 規格 (ISO/IEC 15444-15:2019) のパート 15 で定義されています。HTJ2K は、解像度のスケーラビリティ、プリシンクト、タイリング、高ビット深度、複数チャネル、色空間のサポートなど、JPEG2000 の高度な機能を引き継いでいます。

トピック

- HTJ2K デコーディングライブラリ
- イメージビューワー

HTJ2K デコーディングライブラリ

プログラミング言語に応じて、イメージフレーム をデコードするには、次のデコードライブラリをお勧めします。

- NVIDIA NVJPEG2000 — GPU アクセラレーション対応の商用版
- カカドゥ・ソフトウェア — 商用、Java および .NET バインディングを含む C++
- OpenJPH — オープンソース、C++、WASM
- OpenJPEG — オープンソース、C/C++、Java
- openjphpy — オープンソース、Python
- pylibjpeg-openjpeg — オープンソース、Python

イメージビューワー

イメージフレームは、デコード後に表示できます。AWS HealthImaging API アクションは、次のようなさまざまなオープンソースのイメージビューワーをサポートします。

- Open Health Imaging Foundation (OHIF)

- Cornerstone.js

AWS HealthImaging エンドポイントとクォータ

以下のトピックでは、AWS HealthImaging サービスエンドポイントとクォータについて説明します。

トピック

- サービスエンドポイント
- Service Quotas

サービスエンドポイント

サービスポイントとは、ホストとポートをウェブサービスのエンドポイントとして識別する URL のことです。ウェブサービスの各リクエストには、1 つずつエンドポイントが含まれています。ほとんどの AWS サービスは、特定のリージョンのエンドポイントを提供し、より高速な接続を可能にします。次の表に、AWS のサービスエンドポイントを示します HealthImaging。

リージョン名	リージョン	エンドポイント	プロトコル	
米国東部 (バージニア北部)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS	
米国西部 (オレゴン)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS	
アジアパシフィック (シドニー)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS	
欧州 (アイルランド)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS	

HTTP リクエストを使用して AWS HealthImaging アクションを呼び出す場合は、呼び出されるアクションに応じて異なるエンドポイントを使用する必要があります。次のメニューには、HTTP リクエストで使用可能なサービスエンドポイントと、それらがサポートするアクションがリストされます。

HTTP リクエストでサポートされている API アクション

data store, import, tagging

次のデータストア 、インポート 、およびタグ付けアクションには、エンドポイント経由でアクセスできます。

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- StartDICOMImportJob
- GetDICOMImportJob
- ListDICOMImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

次の画像セットアクションには、エンドポイント経由でアクセスできます。

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging は、DICOMweb 取得 WADO-RS サービスを表します。詳細については、「DICOM インスタンスの取得」を参照してください。

次の DICOMweb サービスにはエンドポイント経由でアクセスできます。

```
https://dicom-medical-imaging.region.amazonaws.com
```

- GetDICOMInstance

Service Quotas

サービスクォータは、 AWS アカウント内のリソース、アクション、および項目の最大値として定義されます。

> **Note**
>
> 調整可能なクォータの場合、Service Quotas コンソールを使用してクォータの引き上げをリクエストできます。詳細については、「Service Quotas ユーザーガイド」の「クォータの引き上げのリクエスト」を参照してください。

次の表に、AWS のデフォルトクォータを示します HealthImaging。

名前	デフォルト	引き上げ可能	説明
データストアあたりの最大同時 CopyImageSet リクエスト数	サポートされている各リージョン: 100	はい	現在の AWS リージョンのデータストアあたりの最大同時 CopyImageSet リクエスト数
データストアあたりの同時 DeleteImageSet リクエストの最大数	サポートされている各リージョン: 100	はい	現在の AWS リージョンのデータストアあたりの最大同時 DeleteImageSet リクエスト数
データストアあたりの同時 UpdateImageSetMetadata リクエストの最大数	サポートされている各リージョン: 100	はい	現在の AWS リージョンのデータストアあたりの最大同時 UpdateImageSetMetadata リクエスト数
データストアあたりの最大同時インポートジョブ	ap-southeast-2: 20 他のサポートされている各リージョン: 100	はい	現在の AWS リージョンのデータストアあたりの同時インポートジョブの最大数

名前	デフォルト	引き上げ可能	説明
最大データストア	サポートされている各リージョン: 10	はい	現在の AWS リージョンのアクティブなデータストアの最大数
CopyImageSet リクエストごとにコピー ImageFrames できる の最大数	サポートされている各リージョン: 1,000	はい	現在の AWS リージョンで CopyImageSet リクエストごとにコピー ImageFrames できる の最大数
DICOM インポートジョブ内のファイルの最大数	サポートされている各リージョン: 5,000	はい	現在の AWS リージョンの DICOM インポートジョブ内のファイルの最大数
DICOM インポートジョブ内の入れ子になったフォルダーの最大数	サポートされている各リージョン: 10,000	いいえ	現在の AWS リージョンの DICOM インポートジョブ内のネストされたフォルダの最大数
で許容される最大ペイロードサイズ制限 (KB) UpdateImageSetMetadata	サポートされている各リージョン: 10 KB	はい	現在の AWS リージョン UpdateImageSetMetadata で が受け入れる最大ペイロードサイズ制限 (KB)
DICOM インポートジョブ内のすべてのファイルの最大サイズ (GB 単位)	サポートされている各リージョン: 10 GB	いいえ	現在の AWS リージョンの DICOM インポートジョブ内のすべてのファイルの最大サイズ (GB 単位)

名前	デフォルト	引き上げ可能	説明
DICOM インポートジョブ内の各 DICOM P10 ファイルの最大サイズ (GB 単位)	サポートされている各リージョン: 4 GB	いいえ	現在の AWS リージョンの DICOM インポートジョブ内の各 DICOM P10 ファイルの最大サイズ (GB 単位)
インポート、コピー、および ImageSetMetadata あたりの の最大サイズ制限 (MB) UpdateImageSet	サポートされている各リージョン: 50 MB	はい	UpdateImageSet 現在の AWS リージョンにおけるインポート、コピー、および ImageSetMetadata あたりの の最大サイズ制限 (MB)

AWS HealthImaging スロットリングの制限

AWS アカウントには、AWS HealthImaging API アクションに適用されるスロットリング制限があります。すべてのアクションで、スロットリング制限を超えた場合に `ThrottlingException` エラーが発生します。詳細については、「AWS HealthImaging API リファレンス」を参照してください。

> **Note**
>
> スロットリング制限は、すべての HealthImaging API アクションで調整できます。スロットリング制限の調整をリクエストするには、AWS サポートセンターにお問い合わせください。

次の表に、AWS HealthImaging API アクションのスロットリング制限を示します。

AWS HealthImaging スロットリングの制限

アクション	スロットリングレート	スロットリングバースト
CreateDatastore	0.085 tps	1 tps
GetDatastore	10 tps	20 tps
ListDatastores	5 tps	10 tps
DeleteDatastore	0.085 tps	1 tps
StartDICOMImportJob	0.25 tps	1 tps
GetDICOMImportJob	25 tps	50 tps
ListDICOMImportJobs	10 tps	20 tps
SearchImageSets	25 tps	50 tps
GetImageSet	25 tps	50 tps
GetImageSetMetadata	50 tps	100 tps
GetImageFrame	1,000 tps	2,000 tps
GetDICOMInstance *	50 tps	100 tps
ListImageSetVersions	25 tps	50 tps
UpdateImageSetMetadata	0.25 tps	1 tps
CopyImageSet	0.25 tps	1 tps
DeleteImageSet	0.25 tps	1 tps
TagResource	10 tps	20 tps
ListTagsForResource	10 tps	20 tps
UntagResource	10 tps	20 tps

*DICOMweb サービスの表現

AWS HealthImaging のサンプルプロジェクト

AWS HealthImagingは GitHub で以下のサンプルプロジェクトを提供しています。

[オンプレミスから AWS HealthImaging への DICOM の取り込み]

DICOM DIMSE ソース (PACS、VNA、CT スキャナー) から DICOM ファイルを受信し、安全な Amazon S3 バケットに保存する、IoT エッジソリューションをデプロイするための AWS サーバーレスプロジェクト。このソリューションは、データベース内の DICOM ファイルのインデックスを作成し、各 DICOM シリーズを AWS HealthImaging にインポートするキューに入れます。これは、AWS IoT Greengrass によって管理されるエッジで実行されるコンポーネントと、AWS クラウドで実行される DICOM 取り込みパイプラインで構成されています。

[タイルレベルマーカー (TLM) プロキシ]

High-Throughput JPEG 2000 (HTJ2K) の機能であるタイルレベルマーカー (TLM) を使用して AWS HealthImaging からイメージフレームを取得する AWS Cloud Development Kit (AWS CDK) プロジェクト。これにより、低解像度の画像では取得時間が短縮されます。考えられるワークフローには、サムネイルの生成や画像の段階的な読み込みなどがあります。

[Amazon CloudFront 配信]

エッジからイメージフレームをキャッシュして (GET を使用) 配信する HTTPS エンドポイントで Amazon CloudFront ディストリビューションを作成するための AWS サーバーレスプロジェクト。デフォルトでは、エンドポイントは Amazon Cognito JSON Web トークン (JWT) を使用してリクエストを認証します。認証とリクエスト署名はどちらも、Lambda @Edge を使用してエッジで行われます。このサービスは Amazon CloudFront の機能で、アプリケーションのユーザーの近くでコードを実行できるため、パフォーマンスが向上し、レイテンシーが減少します。インフラストラクチャを管理する必要はありません。

[AWS HealthImaging ビューワー UI]

AWS HealthImaging に保存されている画像セットのメタデータ属性とイメージフレーム (ピクセルデータ) をプログレッシブデコードで表示できる、バックエンド認証付きのフロントエンド UI をデプロイする AWS Amplify プロジェクト。オプションで、上記のタイルレベルマーカー (TLM) プロキシや Amazon CloudFront 配信プロジェクトを統合して、別の方法でイメージフレームを読み込むことができます。

その他のサンプルプロジェクトを表示するには、GitHub の「AWS HealthImaging のサンプル」を参照してください。

AWS SDK HealthImaging での 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する 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	PowerShell コード例のツール
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 (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

AWS HealthImaging リリース

次の表は、AWS HealthImaging サービスとドキュメントの機能および更新がいつリリースされたかを示しています。

変更	説明	日付

標準以外の DICOM データインポートのサポートを強化	HealthImaging は、DICOM 標準からの逸脱を含むデータインポートをサポートします。詳細については、「DICOM 要素の制約」を参照してください。
	• 次の DICOM データ要素は最大 256 文字です。
	  • `Patient's Name (0010,0010)`
	  • `Patient ID (0010,0020)`
	  • `Accession Number (0008,0050)`
	• 、`Study Instance UID`、、`Series Instance UID`、、`Device UID`および ではTreatment Session UIDManufacturer’s Device Class UID、次の構文バリエーションを使用できますAcquisition UID 。
	  • 任意の UID の最初の要素はゼロにすることができます
		2024 年 6 月 28 日

	• UIDsは 1 つ以上の先頭の 0 から開始できます • UIDsの長さは最大 256 文字です	
イベント通知	HealthImaging は Amazon と統合 EventBridge して、イベント駆動型アプリケーションをサポートします。詳細については、「EventBridge での Amazon の使用 HealthImaging」を参照してください。	2024 年 6 月 5 日
GetDICOMInstance DICOM データを返すための	HealthImaging は、特定の画像セットの DICOM Part 10 データ (.dcm ファイル) を返すGetDICOMInstance サービスを提供します。詳細については、「DICOM インスタンスの取得」を参照してください。	2024 年 5 月 15 日
クロスアカウントインポート	HealthImaging は、サポートされている他のリージョンにある Amazon S3 バケットからのデータインポートをサポートします。詳細については、「のクロスアカウントインポート AWS HealthImaging」を参照してください。	2024 年 5 月 15 日

画像セットの検索機能強化	HealthImaging SearchImageSets アクションは、以下の検索機能強化をサポートしています。詳細については、「画像セットの検索」を参照してください。 • UpdatedAt および の検索に関する追加サポート SeriesInstanceUID • 開始時刻と終了時刻の間の検索 • 検索結果を Ascending または で並べ替える Descending • DICOM シリーズパラメータはレスポンスで返されます	2024 年 4 月 3 日
インポートの最大ファイルサイズの増加	HealthImaging は、インポートジョブ内の DICOM P10 ファイルごとに最大 4 GB のファイルサイズをサポートします。詳細については、「Service Quotas」を参照してください。	2024 年 3 月 6 日

JPEG 可逆および HTJ2K の転送構文	HealthImaging では、ジョブのインポートに次の転送構文がサポートされています。詳細については、「サポートされる転送構文」を参照してください。 • 1.2.840.10008.1.2.4.57 — JPEG 可逆非階層型 (プロセス 14) • 1.2.840.10008.1.2.4.201 — 高スループット JPEG 2000 イメージ圧縮 (ロスレスのみ) • 1.2.840.10008.1.2.4.202 — RPCL オプションによる高スループット JPEG 2000 イメージ圧縮 (ロスレスのみ) • 1.2.840.10008.1.2.4.203 — 高スループット JPEG 2000 イメージ圧縮	2024 年 2 月 16 日
テスト済みのコード例	HealthImaging ドキュメントには、Python、Java JavaScript、C++ 用の および AWS CLI AWS SDKs のテスト済みコード例が記載されています。詳細については、「AWS SDKs HealthImaging を使用するためのコード例」を参照してください。	2023 年 12 月 19 日

インポートの最大ファイル数の増加	HealthImaging は、1 つのインポートジョブで最大 5,000 個のファイルをサポートします。詳細については、「Service Quotas」を参照してください。	2023 年 12 月 19 日
インポート用の入れ子になったフォルダー	HealthImaging は、1 つのインポートジョブに対して最大 10,000 個のネストされたフォルダをサポートします。詳細については、「Service Quotas」を参照してください。	2023 年 12 月 1 日
インポートが速くなります	HealthImaging は、サポートされているすべてのリージョンで 20X高速なインポートを提供します。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。	2023 年 12 月 1 日
CloudFormation サポート	HealthImaging は、データストアをプロビジョニングするための infrastructure as code (IaC) をサポートしています。詳細については、「AWS CloudFormation による AWS HealthImaging リソースの作成」を参照してください。	2023 年 9 月 21 日

一般提供	AWS HealthImaging は、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)、アジアパシフィック (シドニー) の各リージョンのすべてのお客様が利用できます。	2023 年 7 月 26 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。