



デベロッパーガイド

AWS HealthImaging



AWS HealthImaging: デベロッパーガイド

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

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

Table of Contents

AWS HealthImaging とは	1
重要な注意点	2
機能	2
関連サービス	4
アクセス	4
HIPAA	5
料金	5
開始方法	6
概念	6
データストア	6
画像セット	7
メタデータ	7
画像フレーム	7
設定	8
にサインアップする AWS アカウント	8
管理アクセスを持つユーザーを作成する	9
S3 バケットを作成する	10
データストアの作成	11
IAM ユーザーの作成	11
IAM ロールを作成する	12
のインストール AWS CLI	14
チュートリアル	15
データストアの管理	16
データストアの作成	16
データストアのプロパティの取得	25
データストアの一覧表示	33
データストアの削除	41
DICOMweb の使用	48
STOW-RS を使用したインスタンスの保存	48
WADO-RS を使用したデータの取得	52
インスタンスを取得する	53
インスタンスメタデータの取得	56
シリーズメタデータを取得する	57
フレームを取得する	58

バルクデータを取得する	61
"DO-RS を使用したデータの検索	63
HealthImaging の DICOMweb 検索 APIs	63
HealthImaging でサポートされている DICOMweb クエリタイプ	64
検査を検索する	68
シリーズの検索	70
インスタンスの検索	71
OIDC 認証	72
トークン検証の仕組み	73
要件とセットアップ	77
画像データのインポート	88
インポートジョブを理解する	88
インポートジョブの開始	92
インポートジョブプロパティの取得	101
インポートジョブの一覧表示	108
画像セットへのアクセス	115
画像セットの理解	115
画像セットの検索	122
画像セットのプロパティの取得	149
画像セットメタデータの取得	155
画像セットのピクセルデータの取得	166
画像セットの変更	175
画像セットのバージョンを一覧表示する	175
画像セットメタデータの更新	182
プライマリイメージセットのメタデータを更新するには	202
プライマリ以外のイメージセットをプライマリにするには	203
画像セットのコピー	204
画像セットの削除	220
リソースのタグ付け	227
リソースのタグging	227
リソースのタグを一覧表示します	233
リソースのタグを削除します	238
コードの例	245
基本	246
Hello HealthImaging	247
アクション	252

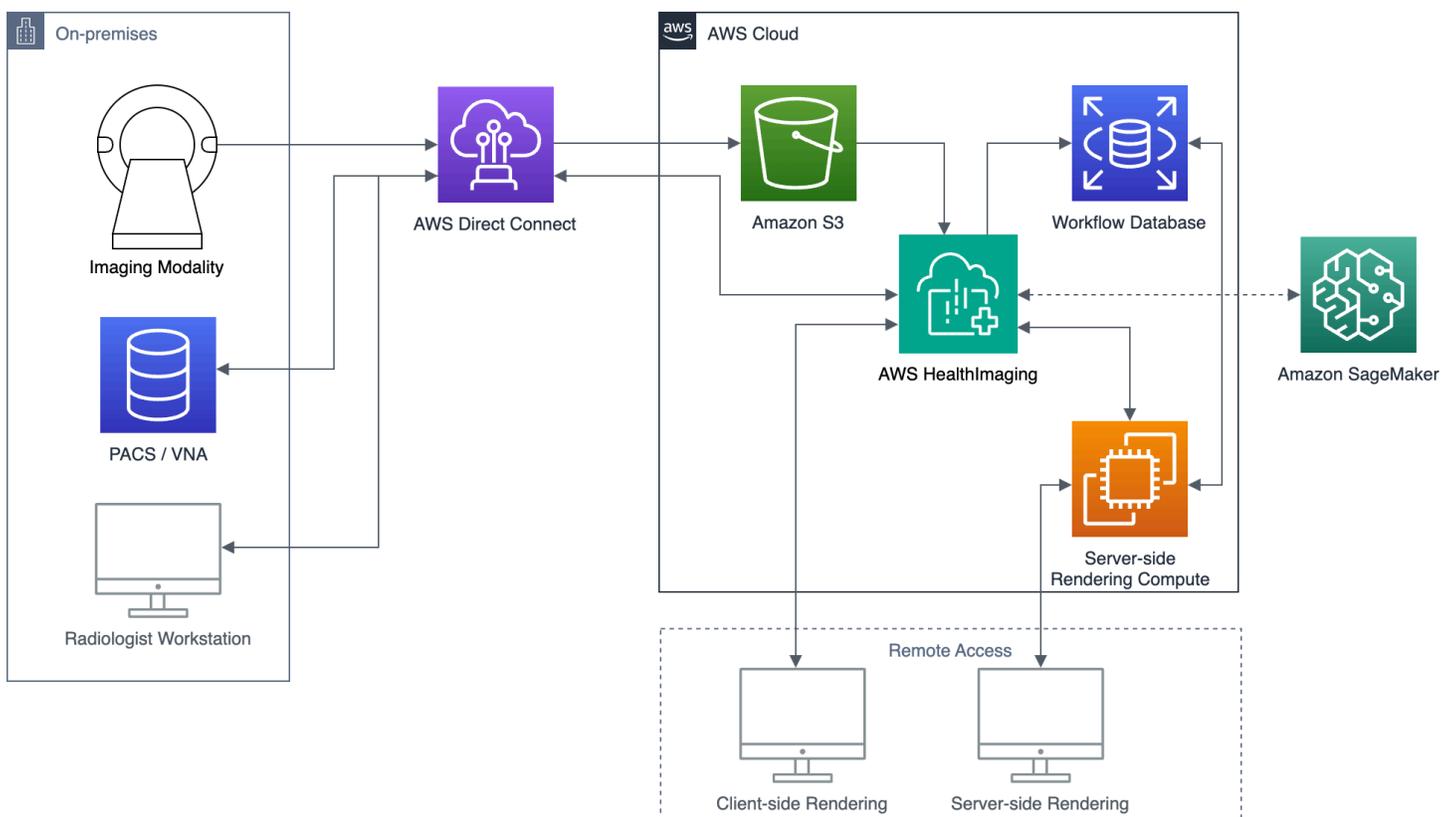
シナリオ	407
画像セットと画像フレームを使い始める	408
データストアにタグを付ける	462
イメージセットにタグを付ける	472
モニタリング	484
CloudTrail (API コール)	485
証跡の作成	485
ログエントリについて	487
CloudWatch (メトリクス)	488
HealthImaging メトリクスの表示	489
アラームを作成する	489
EventBridge (イベント)	490
EventBridge に送信される HealthImaging イベント EventBridge	490
HealthImaging イベント構造と例	491
セキュリティ	508
データ保護	509
データ暗号化	510
ネットワークトラフィックのプライバシー	520
アイデンティティとアクセス管理	520
オーディエンス	521
アイデンティティを使用した認証	521
ポリシーを使用したアクセスの管理	522
AWS HealthImaging が IAM で機能する仕組み	524
アイデンティティベースのポリシーの例	531
AWS マネージドポリシー	534
サービス間の混乱した代理の防止	536
トラブルシューティング	537
コンプライアンス検証	539
インフラストラクチャセキュリティ	540
Infrastructure as Code	540
HealthImaging と CloudFormation テンプレート	541
の詳細 CloudFormation	541
VPC エンドポイント	541
VPC エンドポイントに関する考慮事項	542
VPC エンドポイントの作成	542
VPC エンドポイントポリシーの作成	543

クロスアカウントインポート	544
耐障害性	546
リファレンス	547
DICOM	547
サポートされている SOP クラス	548
メタデータの正規化	548
サポートされる転送構文	553
DICOM 要素の制約	556
DICOM メタデータの制約	557
HealthImaging	557
エンドポイントとクォータ	558
スロットリングの制限	564
ピクセルデータ検証	566
警告コード	568
イメージフレームデコードライブラリ	571
サンプルプロジェクト	572
AWS SDKs の使用	574
コスト最適化	576
インテリジェント階層化の仕組み	576
構造化データストレージの見積もり	577
リリース	579
.....	dx civ

AWS HealthImaging とは

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 対応サービスです。HealthImaging のユースケースは次のとおりです。

- エンタープライズ画像 – 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してストリーミングします。
- 長期イメージアーカイブ – 1 秒未満のイメージ取得アクセスを維持しながら、長期イメージ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、精度医学に関するインサイトを提供します。



AWS HealthImaging は、画像データ (X-Ray、CT、MRI、Ultrasound など) へのアクセスを提供するため、クラウド上に構築された医療画像アプリケーションは、以前はオンプレミスでのみ可能なパ

パフォーマンスを実現できます。HealthImaging では、AWS クラウド内の各医療画像の 1 つの信頼できる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のコストを削減できます。

トピック

- [重要な注意点](#)
- [AWS HealthImaging の機能](#)
- [関連 AWS サービス](#)
- [AWS HealthImaging へのアクセス](#)
- [HIPAA の適格性とデータセキュリティ](#)
- [料金](#)

重要な注意点

AWS HealthImaging は、専門家による医療の助言、診断や治療の代用品ではなく、疾患や健康状態の治療、軽減、予防、診断を目的としたものではありません。AWS HealthImaging の使用の一環として人間による審査を実施する責任はユーザーにあります。これには、臨床上の意思決定の情報提供を目的としたサードパーティ製品に関連するものも含まれます。AWS HealthImaging は、訓練を受けた医療専門家による健全な医療判断の確認後、患者ケアのシナリオまたは臨床シナリオでのみ使用してください。

AWS HealthImaging の機能

AWS HealthImaging には、次の機能があります。

開発者に使いやすい DICOM メタデータ

AWS HealthImaging は、DICOM メタデータを開発者にわかりやすい形式で返すことで、アプリケーション開発を簡素化します。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 HealthImaging は、高度な画像圧縮コーデックである高スループット JPEG 2000 (HTJ2K) 画像としてエンコードされた画像フレーム (ピクセルデータ) 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。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。

ピクセルデータ検証

AWS HealthImaging には、インポート中にすべての画像の可逆的エンコードとデコード状態をチェックすることで、ピクセルデータ検証が組み込まれています。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 HealthImaging 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、画像の読み込みパフォーマンスの新しい基準を打ち立てました。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 のインポートは、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 治験を同時にインポートします。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](#)」を参照してください。

サービスマネージド DICOM データ階層

AWS HealthImaging は、インポート時に患者、治験、シリーズレベルの DICOM データ要素によって DICOM P10 データを自動的に整理します。このサービスは、この DICOM データを DICOM シリーズに対応する画像セットに整理し、インポート後のワークフローを簡素化します。新しいデータがインポートされると、治験レベルとシリーズレベルの組織は維持されます。

DICOMweb API の互換性

AWS HealthImaging は DICOMweb 準拠 APIs を提供し、統合を簡素化し、既存のアプリケーションとの相互運用性を実現します。このサービスは、メタデータ更新オペレーションな

ど、DICOMweb 標準でサポートされていないアクションを有効にするクラウドネイティブ APIs も提供します。

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。HealthImaging を最大限に活用するには、以下のサービスに関する知識が役立ちます。

- [AWS Identity and Access Management](#) – IAM を使用して、アイデンティティと 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 HealthImaging にインポートします。
- [Amazon CloudWatch](#) – CloudWatch を使用して HealthImaging リソースを監視およびモニタリングします。
- [AWS CloudTrail](#) – CloudTrail を使用して HealthImaging ユーザーアクティビティと API 使用状況を追跡します。
- [AWS CloudFormation](#) – CloudFormation を使用して、Infrastructure as Code (IaC) テンプレートを実装し、HealthImaging でリソースを作成します。
- [AWS PrivateLink](#) – Amazon VPC を使用して、インターネットにデータを公開することなく HealthImaging と [Amazon Virtual Private Cloud](#) 間の接続を確立します。
- [Amazon EventBridge](#) – EventBridge を使用して、HealthImaging 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して、スケーラブルなイベント駆動型アプリケーションを作成します。

AWS HealthImaging へのアクセス

AWS Command Line Interface および AWS SDKs を使用して AWS マネジメントコンソール AWS HealthImaging にアクセスできます。このガイドでは、 の手順と、 AWS マネジメントコンソール および AWS CLI SDKs の AWS コード例について説明します。

AWS マネジメントコンソール

AWS マネジメントコンソール は、HealthImaging とその関連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、「[構築するツール AWS](#)」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ができますが、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。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 サービスを使用した保護医療情報 (PHI) の処理、保存、および送信の詳細については、「[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) を含む HealthImaging への接続は暗号化する必要があります。デフォルトでは、HealthImaging へのすべての接続で HTTPS over TLS を使用します。HealthImaging は暗号化された顧客コンテンツを保存し、[AWS 責任分担モデル](#)に従って運営されています。

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、「[コスト最適化](#)」を参照してください。

一般的な料金情報については、「[AWS HealthImaging 料金表](#)」を参照してください。費用を見積もるには、[AWS HealthImaging 料金計算ツール](#)を使用してください。

AWS HealthImaging の開始方法

AWS HealthImaging の使用を開始するには、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 の概念と設定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 HealthImaging の設定](#)
- [AWS HealthImaging のチュートリアル](#)

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 AWS リージョンにある医療画像データのリポジトリです。AWS アカウントには、ゼロまたは多数のデータストアを含めることができます。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「[コスト最適化](#)」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。HealthImaging は、インポートされたデータを、治験、シリーズ、インスタンスの DICOM 階層に従って整理しようとしています。HealthImaging 管理階層に正常に追加された DICOM インスタンスは、プライマリ[イメージセット](#)として表示されます。DICOM P10 データをインポートするには、新しいプライマリ[イメージセット](#)を作成するか、インスタンスがプライマリコレクションに既に存在する場合はインスタンスを既存のプライマリ[イメージセット](#)にマージするか、メタデータ要素が競合する場合は新しい非プライマリ[イメージセット](#)を作成します。

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。一部のファイルはインポート中に元の転送構文エンコードを保持し、他のファイルはトランスコードされます。Amazon Web Services データストアは、可逆画像フレームを高スループット JPEG 2000 (HTJ2K) 可逆または JPEG 2000 可逆に変換するように設定できます。イメージフレームが HTJ2K または JPEG 2000 でエンコードされている場合は、イメージビューア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[サポートされる転送構文](#)、[画像セットのピクセルデータの取得](#)、および[イメージフレームデコードライブラリ](#)を参照してください。

AWS HealthImagingの設定

AWS HealthImaging を使用する前に AWS 環境を設定する必要があります。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 の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、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

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

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

管理アクセスを持つユーザーを作成する

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

を保護する AWS アカウントのルートユーザー

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

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

2. ルートユーザーの多要素認証 (MFA) を有効にします。

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

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

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

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

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

管理アクセス権を持つユーザーとしてサインインする

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

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

追加のユーザーにアクセス権を割り当てる

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

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

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

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

S3 バケットを作成する

DICOM P10 データを AWS HealthImaging にインポートするには、2 つの Amazon S3 バケットを使用することをお勧めします。Amazon S3 入力バケットにはインポートする DICOM P10 データが保存され、HealthImaging はこのバケットからデータを読み取ります。Amazon S3 出力バケットにはインポートジョブの処理結果が保存され、HealthImaging は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:::amzn-s3-demo-source-bucket`
- 出力バケット: `arn:aws:s3:::amzn-s3-demo-logging-bucket`

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](#)に変換済み DICOM P10 ファイルの結果が格納されます。これらは[画像セット](#)と呼ばれます。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。

HealthImafig の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 AWS HealthImaging の 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 AWS HealthImaging の 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もう 1 つの 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。AWS HealthImaging がロールを引き受けられるように、信頼関係 (ポリシー) も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。

JSON

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

```
{
  "Action": [
    "s3:ListBucket"
  ],
  "Resource": [
    "arn:aws:s3:::amzn-s3-demo-source-bucket",
    "arn:aws:s3:::amzn-s3-demo-logging-bucket"
  ],
  "Effect": "Allow"
},
{
  "Action": [
    "s3:GetObject"
  ],
  "Resource": [
    "arn:aws:s3:::amzn-s3-demo-source-bucket/*"
  ],
  "Effect": "Allow"
},
{
  "Action": [
    "s3:PutObject"
  ],
  "Resource": [
    "arn:aws:s3:::amzn-s3-demo-logging-bucket/*"
  ],
  "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。このトピックに続く [チュートリアル](#) では、1 つの AWS HealthImaging データストアの使用を想定していますが、データストア固有の Amazon S3 バケット、IAM ロール、信頼ポリシーがあります。

Note

この信頼ポリシーの Condition ブロックは、特定の AWS HealthImaging データストアにのみアクセスできるようにすることで、混乱した代理問題を防ぐのに役立ちます。こ

のセキュリティ対策の詳細については、[HealthImaging でのサービス間の混乱した代理の防止](#)を参照してください。

AWS HealthImaging で IAM ポリシーを作成・使用する詳しい方法については、[AWS HealthImaging のアイデンティティとアクセス管理](#)を参照してください。

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 マネジメントコンソール 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。

をセットアップするには 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。AWS CLI コマンドを実行するときは、このプロファイルを使用します。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 のチュートリアル

目的

このチュートリアルの目的は、DICOM P10 バイナリ (.dcm 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後、HealthImaging クラウドネイティブアクションを使用して、[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。

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 HealthImaging によるデータストアの管理

AWS HealthImaging を使用して、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、HealthImaging クラウドネイティブアクションを使用して、`awscli`、`awscli`、AWS SDK を使用してデータストアを作成、説明 AWS マネジメントコンソール、一覧表示 AWS CLI、削除する方法について説明します。 SDKs

Note

この章の最後のトピックでは、[コストの最適化](#)について説明します。HealthImaging データストアに医療画像データをインポートすると、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](#)
- [データストアのプロパティの取得](#)
- [データストアの一覧表示](#)
- [データストアの削除](#)

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、`awscli` の手順 AWS マネジメントコンソール と、AWS CLI および 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。データストアを作成するときに、AWS HealthImaging が可逆画像フレームのトランスコードと保存に使用したデフォルトの転送構文を選択できます。データストアの作成後にこの設定を変更することはできません。

高スループット JPEG 2000 (HTJ2K)

HTJ2K (高スループット JPEG 2000) は、HealthImaging データストアのデフォルトのストレージ形式です。これは JPEG 2000 標準の拡張であり、エンコードとデコードのパフォーマンスが大幅に向上します。を指定せずにデータストアを作成すると`-lossless-storage-format`、HealthImaging は自動的に HTJ2K を使用します。HTJ2K SDKs セクションを参照してください。 HTJ2K

JPEG 2000 ロスレス

JPEG 2000 ロスレスエンコーディングを使用すると、トランスコードなしで JPEG 2000 形式でロスレスイメージフレームを保持および取得するデータストアを作成できるため、JPEG 2000 ロスレス (DICOM Transfer Syntax UID 1.2.840.10008.1.2.4.90) を必要とするアプリケーションのレイテンシーを短縮できます。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。JPEG 2000 ロスレス形式を使用したデータストアの作成については、以下の AWS CLI および SDKs セクション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。
- AWS コンソールは、デフォルト設定でのデータストアの作成をサポートしています。CLI または AWS SDK AWS を使用して、オプションの `lossless-storage-format` を指定してデータストアを作成します。

データストアを作成するには

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

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 HealthImaging でのデータ保護](#)」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、「[リソースのタグging](#)」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 getopt 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、AWS CLI コマンドリファレンスの「[CreateDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

CLI

AWS CLI

例 1: データストアにタグを付けるには

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。を指定せずにデータストアを作成する場合 --lossless-storage-format、AWS HealthImaging のデフォルトは HTJ2K (高スループット JPEG 2000) です。

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

出力:

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

例 2: JPEG 2000 ロスレスストレージ形式でデータストアを作成するには

JPEG 2000 ロスレスストレージ形式で設定されたデータストアは、可逆イメージフレームを JPEG 2000 形式でトランスコードして保持します。その後、画像フレームは JPEG 2000 ロスレス、トランスコーディングなしで取得できます。次の create-datastore コード例では、my-datastore という名前の JPEG 2000 ロスレスストレージ形式用に設定されたデータストアを作成します。

```
aws medical-imaging create-datastore \  
  --datastore-name "my-datastore" \  
  --lossless-storage-format HTJ2K
```

```
--lossless-storage-format JPEG_2000_LOSSLESS
```

出力:

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

詳細については、「AWS HealthImaging Developer Guide」の「[Creating a data store](#)」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「[CreateDatastore](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[CreateDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'CREATING'
  // }
  return response;
};
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[CreateDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_datastore_name = 'my-datastore-name'  
    oo_result = lo_mig->createdatastore( iv_datastorename =  
iv_datastore_name ).  
    DATA(lv_datastore_id) = oo_result->get_datastoreid( ).  
    MESSAGE 'Data store created.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
    MESSAGE 'Conflict. Data store may already exist.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migservicequotaexcdex.  
    MESSAGE 'Service quota exceeded.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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 AWS マネジメントコンソール と、 AWS CLI および 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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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 getopt 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 getopt "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、「AWS CLI コマンドリファレンス」の「[GetDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

CLI

AWS CLI

例 1: 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",  
    "losslessStorageFormat": "HTJ2K"  
    "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"  
  }  
}
```

例 2: JPEG 2000 用に設定された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、JPEG 2000 ロスレスストレージ形式用に設定されたデータストアのプロパティを取得します。

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

出力:

```
{
  "datastoreProperties": {
    "datastoreId": "12345678901234567890123456789012",
    "datastoreName": "TestDatastore123",
    "datastoreStatus": "ACTIVE",
    "losslessStorageFormat": "JPEG_2000_LOSSLESS",
    "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 Developer Guide」の「[Getting data store properties](#)」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「[GetDatastore](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[GetDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for JavaScript API リファレンスの「[GetDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[GetDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->getdatastore( iv_datastoreid = iv_datastore_id ).
    DATA(lo_properties) = oo_result->get_datastoreproperties( ).
    DATA(lv_name) = lo_properties->get_datastorename( ).
    DATA(lv_status) = lo_properties->get_datastorestatus( ).
    MESSAGE 'Data store properties retrieved.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Data store not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
```

```
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの[GetDatastore](#) を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

データストアの一覧表示

ListDatastores アクションを使用して、AWS HealthImaging で使用可能な[データストア](#)を一覧表示します。次のメニューでは、 の手順 AWS マネジメントコンソール と、AWS CLI および 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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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 getopt 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 getopt "h" option; do
        case "${option}" in
            h)
                usage
                return 0
                ;;
        esac
    done
}
```

```
\?)
    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、AWS CLI コマンドリファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Listing data stores](#)」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「[ListDatastores](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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);
    }
}
```


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、AWS SDK for Python (Boto3) API リファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    oo_result = lo_mig->listdatastores( ).  
    DATA(lt_datastores) = oo_result->get_datastoresummaries( ).  
    DATA(lv_count) = lines( lt_datastores ).  
    MESSAGE |Found { lv_count } data stores.| TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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 AWS マネジメントコンソール と、 AWS CLI および 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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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 getopt 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 getopt "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、「AWS CLI コマンドリファレンス」の「[DeleteDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Deleting a data store](#)」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「[DeleteDatastore](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[DeleteDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'DELETING'
  // }

  return response;
};
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[DeleteDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_datastore_id = '1234567890123456789012345678901234567890'  
    oo_result = lo_mig->deletedatastore( iv_datastoreid = iv_datastore_id ).  
    MESSAGE 'Data store deleted.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
    MESSAGE 'Conflict. Data store may contain resources.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Data store not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

AWS HealthImaging での DICOMweb の使用

DICOMweb APIs の表現を使用して AWS HealthImaging から [DICOMweb](#) API は、医療画像用の DICOM 標準に準拠したウェブベースの APIs です。この機能を使用すると、HealthImaging の [クラウドネイティブアクション](#) 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。この章では、HealthImaging の DICOMweb APIs を返す方法について説明します。

DICOMweb

[重要]

HealthImaging は DICOM データを [画像セット](#) として保存します。HealthImaging クラウドネイティブアクションを使用して、イメージセットを管理および取得します。HealthImaging の DICOMweb APIs を使用して、DICOMweb 準拠のレスポンスで画像セット情報を返すことができます。

この章に記載されている APIs は、ウェブベースの医療画像用の [DICOMweb](#) 標準に準拠して構築されています。これらは DICOMweb APIs の表現であるため、AWS CLI および AWS SDKs では提供されません。

Topic

- [STOW-RS を使用したインスタンスの保存](#)
- [HealthImaging から DICOM データを取得する](#)
- [HealthImaging での DICOM データの検索](#)
- [DICOMweb APIs の OIDC 認証](#)

STOW-RS を使用したインスタンスの保存

AWS HealthImaging は、データをインポートするための [DICOMweb STOW-RS](#) APIs の表現を提供します。これらの APIs を使用して、DICOM データを HealthImaging データストアに同期的に保存します。

次の表は、データのインポートに使用できる DICOMweb STOW-RS APIs の HealthImaging 表現を示しています。

DICOMweb STOW-RS APIs の HealthImaging 表現

名前	説明
StoreDICOM	HealthImaging データストアに 1 つ以上のインスタンスを保存します。
StoreDICOMStudy	指定された治験インスタンス UID に対応する 1 つ以上のインスタンスを HealthImaging データストアに保存します。

StoreDICOM および StoreDICOMStudy アクションでインポートされたデータは、非同期 [インポートジョブ](#) と同じロジックを使用して、新しいプライマリイメージセットとして編成されるか、既存のプライマリイメージセットに追加されます。新しくインポートされた DICOM P10 データのメタデータ要素が既存のプライマリ [イメージセット](#) と競合する場合、新しいデータは非プライマリ [イメージセット](#) に追加されます。

 Note

- これらのアクションは、リクエストごとに最大 1GB の DICOM データのアップロードをサポートします。
- API レスポンスは、DICOMweb STOW-RS 標準に準拠した JSON 形式になります。

StoreDICOM リクエストを開始するには

1. AWS リージョン、HealthImaging datastoreId、DICOM P10 ファイル名を収集します。
2. フォームのリクエストの URL を作成します。 `https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/studies`
3. などの任意のコマンドを使用して、DICOM P10 ファイルの内容の長さを決定します `$(stat -f %z $FILENAME)`。
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 POST リクエスト StoreDICOM を使用します。

Example例 1: StoreDICOMアクションを使用して DICOM P10 ファイルを保存するには**Shell**

```
curl -X POST -v \  
  'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/  
d9a2a515ab294163a2d2f4069eed584c/studies' \  
  --aws-sigv4 "aws:amz:$AWS_REGION:medical-imaging" \  
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \  
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \  
  --header "x-amz-content-sha256: STREAMING-AWS4-HMAC-SHA256-PAYLOAD" \  
  --header "x-amz-decoded-content-length: $CONTENT_LENGTH" \  
  --header 'Accept: application/dicom+json' \  
  --header "Content-Type: application/dicom" \  
  --upload-file $FILENAME
```

Example例 2: StoreDICOMStudyアクションを使用して DICOM P10 ファイルを保存するには

StoreDICOM と StoreDICOMStudy の唯一の違いは、治験インスタンス UID が StoreDICOMStudy のパラメータとして渡され、アップロードされたインスタンスが指定された治験のメンバーであることです。

Shell

```
curl -X POST -v \  
  'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/  
d9a2a515ab294163a2d2f4069eed584c/studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457'  
 \  
  --aws-sigv4 "aws:amz:$AWS_REGION:medical-imaging" \  
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \  
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \  
  --header "x-amz-content-sha256: STREAMING-AWS4-HMAC-SHA256-PAYLOAD" \  
  --header "x-amz-decoded-content-length: $CONTENT_LENGTH" \  
  --header 'Accept: application/dicom+json' \  
  --header "Content-Type: application/dicom" \  
  --upload-file $FILENAME
```

Example例 3: マルチパート HTTP ペイロードで DICOM P10 ファイルを保存するには

1つのマルチパートアップロードアクションで複数の P10 ファイルをアップロードできます。次のシェルコマンドは、2つの P10 ファイルを含むマルチパートペイロードをアSEMBルし、StoreDICOMアクションでアップロードする方法を示しています。

Shell

```
#!/bin/sh
FILENAME=multipart.payload
BOUNDARY=2a8a02b9-0ed3-c8a7-7ebd-232427531940
boundary_str="--$BOUNDARY\r\n"
mp_header="${boundary_str}Content-Type: application/dicom\r\n\r\n"

##Encapsulate the binary DICOM file 1.
printf '%b' "$mp_header" > $FILENAME
cat file1.dcm >> $FILENAME

##Encapsulate the binary DICOM file 2 (note the additional CRLF before the part
header).
printf '%b' "\r\n$mp_header" >> $FILENAME
cat file2.dcm >> $FILENAME

## Add the closing boundary.
printf '%b' "\r\n--$BOUNDARY--" >> $FILENAME

## Obtain the payload size in bytes.
multipart_payload_size=$(stat -f%z "$FILENAME")

# Execute CURL POST request with AWS SIGv4
curl -X POST -v \
  'https://iad-dicom.external-healthlake-imaging.ai.aws.dev/datastore/
b5f34e91ca734b39a54ac11ea42416cf/studies' \
  --aws-sigv4 "aws:amz:us-east-1:medical-imaging" \
  --user "AKIAIOSFODNN7EXAMPLE:wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY" \
  --header "x-amz-content-sha256: STREAMING-AWS4-HMAC-SHA256-PAYLOAD" \
  --header "x-amz-decoded-content-length: ${multipart_payload_size}" \
  --header 'Accept: application/dicom+json' \
  --header "Content-Type: multipart/related; type=\"application/dicom\"; boundary=
\"${BOUNDARY}\"" \
  --data-binary "@$FILENAME"

# Delete the payload file
```

rm \$FILENAME

HealthImaging から DICOM データを取得する

AWS HealthImaging は、シリーズレベルとインスタンスレベルでデータを取得するための [DICOMweb WADO-RS](#) APIs の表現を提供します。これらの APIs を使用すると、HealthImaging [データストア](#) から DICOM シリーズのすべてのメタデータを取得できます。DICOM インスタンス、DICOM インスタンスメタデータ、DICOM インスタンスフレーム (ピクセルデータ) を取得することもできます。HealthImaging の DICOMweb WADO-RS APIs は、HealthImaging に保存されているデータを取得する方法に柔軟性を提供し、レガシーアプリケーションとの相互運用性を提供します。

[重要]

HealthImaging は、DICOM データを [画像セット](#) として保存します。HealthImaging [クラウドネイティブアクション](#) を使用して、イメージセットを管理および取得します。HealthImaging の DICOMweb APIs を使用して、DICOMweb 準拠のレスポンスで画像セット情報を返すことができます。

このセクションに記載されている APIs は、ウェブベースの医療画像用の DICOMweb (WADO-RS) 標準に準拠して構築されています。これらは DICOMweb APIs の表現であるため、AWS CLI および AWS SDKs では提供されません。

次の表は、HealthImaging からデータを取得するために使用できる DICOMweb WADO-RS APIs のすべての HealthImaging 表現を示しています。

DICOMweb WADO-RS APIs の HealthImaging 表現

名前	説明
GetDICOMSeriesMetadata	リソースに関連付けられた Study UID と Series UIDs を指定して、HealthImaging データストア内の DICOM シリーズの DICOM インスタンスメタデータ (.json ファイル) を取得します。「 シリーズメタデータを取得する 」を参照してください。
GetDICOMInstance	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

名前	説明
GetDICOMInstanceMetadata	<p>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。「インスタンスを取得する」を参照してください。</p> <p>リソースに関連付けられたシリーズ、スタディ、インスタンス UUIDs を指定して、HealthImaging データストアの DICOM インスタンスから DICOM インスタンスメタデータ (.json ファイル) を取得します。「インスタンスメタデータの取得」を参照してください。</p>
GetDICOMInstanceFrames	<p>リソースに関連付けられたシリーズ UID、治療 UID、インスタンス UUIDs、フレーム番号を指定して、HealthImaging データストアの DICOM インスタンスから単一またはバッチイメーجزフレーム (multipart リクエスト) を取得します。「フレームを取得する」を参照してください。</p>

トピック

- [HealthImaging から DICOM インスタンスを取得する](#)
- [HealthImaging からの DICOM インスタンスメタデータの取得](#)
- [HealthImaging からの DICOM シリーズメタデータの取得](#)
- [HealthImaging からの DICOM インスタンスフレームの取得](#)
- [HealthImaging からの DICOM バルクデータの取得](#)

HealthImaging から DICOM インスタンスを取得する

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、治療、インスタンス UUIDs 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。API は、オプションのイメージセット [パラメータが指定されていない限り、プライマリイメージセット](#)からのみインスタンスを返します。をクエリパラメータ imageSetId として指定する

ことで、データストア内の任意のインスタンス (プライマリまたは非プライマリイメージセットから) を取得できます。DICOM データは、保存された転送構文または非圧縮 (ELE) 形式で取得できます。

DICOM インスタンスを取得するには (.dcm)

1. HealthImaging datastoreIdとimageSetId/パラメータ値を収集します。
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、studyInstanceUID、およびの値を使用してsopInstanceUID、リクエストの URL seriesInstanceUIDを作成しますimageSetId。次の例の URL パス全体を表示するには、コピーボタンにスクロールします。URL は 形式です。

```
GET 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。は署名[AWS 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 HealthImaging から DICOM インスタンス (.dcm ファイル) を取得します。

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 はオプションであり、含まれていない場合はデフォルトで Explicit VR Little Endian 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 (可逆画像フレームのデフォルト)
- transfer-syntax=* その場合、イメージフレーム (複数可) は保存された転送構文で返されます。
- 高スループット JPEG 2000 with RPCL Options Image Compression (Lossless Only) - 1.2.840.10008.1.2.4.202 - インスタンスが HealthImaging に次のように保存されている場合 1.2.840.10008.1.2.4.202
- JPEG 2000 可逆 - 1.2.840.10008.1.2.4.90 - インスタンスが可逆として HealthImaging に保存されている場合。
- JPEG ベースライン (プロセス 1): 損失 JPEG 8 ビットイメージ圧縮のデフォルトの転送構文 -- 1.2.840.10008.1.2.4.50 インスタンスが HealthImaging にとして保存されている場合 1.2.840.10008.1.2.4.50
- JPEG 2000 Image Compression - 1.2.840.10008.1.2.4.91 - インスタンスが HealthImaging に次のように保存されている場合 1.2.840.10008.1.2.4.91
- 高スループット JPEG 2000 イメージ圧縮 - 1.2.840.10008.1.2.4.203 - インスタンスが HealthImaging にとして保存されている場合 1.2.840.10008.1.2.4.203
- JPEG XL イメージ圧縮 - 1.2.840.10008.1.2.4.112 - インスタンスが HealthImaging に次のように保存されている場合 1.2.840.10008.1.2.4.112
- HealthImaging に保存されているインスタンスは、[転送構文](#)の MPEG ファミリー (MPEG2, MPEG-4 AVC/H.264、HEVC/H.265 を含む) でエンコードされた 1 つ以上のイメージフレームを使用して、対応する転送構文 UID で取得できます。たとえば、インスタンスが MPEG2 Main Profile Main Level として保存 1.2.840.10008.1.2.4.100 されている場合です。

詳細については、「[サポートされる転送構文](#)」および「[AWS HealthImaging の画像フレームデコードライブラリ](#)」を参照してください。

HealthImaging からの DICOM インスタンスメタデータの取得

GetDICOMInstanceMetadata 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、治療、インスタンス UUIDs を指定して、HealthImaging [データストア](#) の DICOM インスタンスからメタデータを取得します。API は、オプションのイメージセットパラメータが指定されていない限り、プライマリ [イメージセット](#) からのみインスタンスメタデータを返します。をクエリパラメータ imageSetId として指定することで、データストア内の任意のインスタンスメタデータ (プライマリまたは非プライマリイメージセットから) を取得できます。

DICOM インスタンスメタデータを取得するには (.json)

1. HealthImaging datastoreId と imageSetId パラメータ値を収集します。
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、studyInstanceUID、および の値を使用して sopInstanceUID、リクエストの URL seriesInstanceUID を作成します imageSetId。次の例の URL パス全体を表示するには、コピーボタンにスクロールします。URL は 形式です。

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

4. リクエストを準備して送信します。は署名 [AWS バージョン 4](#) の署名プロトコルで HTTP GET リクエスト GetDICOMInstanceMetadata 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 HealthImaging から DICOM インスタンスメタデータ (.json ファイル) を取得します。

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/metadata?  
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+json'
```

Note

メタデータに示されている転送構文 UID は、HealthImaging のストア転送構文 UID (StoredTransferSyntaxUID) と一致します。

HealthImaging からの DICOM シリーズメタデータの取得

GetDICOMSeriesMetadata アクションを使用して、HealthImaging [データストア](#) から DICOM シリーズ (.json ファイル) のメタデータを取得します。リソースに関連付けられた Study および Series UUIDs を指定することで、HealthImaging データストア内の任意のプライマリ [イメージセット](#) のシリーズメタデータを取得できます。非プライマリイメージセットのシリーズメタデータを取得するには、イメージセット ID をクエリパラメータとして指定します。シリーズメタデータは DICOM JSON形式で返されます。

DICOM シリーズメタデータを取得するには (.json)

1. HealthImaging datastoreId と imageSetId パラメータ値を収集します。
2. datastoreId、studyInstanceUID、およびオプションで の値を使用して seriesInstanceUID、リクエストの URL を作成します imageSetId。次の例の URL パス全体を表示するには、コピーボタンをスクロールします。URL は 形式です。

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/  
studies/study-instance-uid/series/series-instance-uid/metadata
```

3. リクエストを準備して送信します。は署名 [AWS バージョン 4](#) の署名プロトコルで HTTP GET リクエスト GetDICOMSeriesMetadata 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 HealthImaging からメタデータ (.json ファイル) を取得します。

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/metadata \  
--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+json' \  
--output 'series-metadata.json'
```

オプションの `imageSetId` パラメータを使用。

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/metadata?  
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+json' \  
--output 'series-metadata.json'
```

Note

- `imageSetId` パラメータは、非プライマリイメージセットのシリーズメタデータを取得するために必要です。GetDICOMInstanceMetadata アクションは、`datastoreId`、`seriesInstanceUID` が指定されている (なし) 場合にのみ `studyInstanceUID`、プライマリイメージセットのシリーズメタデータを返します `imagesetID`。

HealthImaging からの DICOM インスタンスフレームの取得

GetDICOMInstanceFrames アクションを使用して、HealthImaging [データストア](#) の DICOM インスタンスから単一またはバッチイメージフレーム (multipart リクエスト) を取得するには、リソースに関連付けられたシリーズ UID、治験 UID、インスタンス UIDs、フレーム番号を指定します。 [1](#)

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、インスタンスフレームを取得するイメージセットを指定できます。API は、オプションのイメージセットパラメータが指定されていない限り、プライマリ**イメージセット**からのインスタンスフレームのみを返します。をクエリパラメータ `imageSetId` として指定することで、データストア内の任意のインスタンスフレーム (プライマリまたは非プライマリイメージセットから) を取得できます。

DICOM データは、保存された転送構文または非圧縮 (ELE) 形式で取得できます。

DICOM インスタンスフレームを取得するには (**multipart**)

1. HealthImaging `datastoreId` と `imageSetId` パラメータ値を収集します。
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. 関連付けられたメタデータから取得するイメージフレームを決定して、`frameList` パラメータを形成します。`frameList` パラメータは、任意の順序で 1 つ以上の重複しないフレーム番号のカンマ区切りリストです。たとえば、メタデータの最初のイメージフレームはフレーム 1 になります。
 - 単一フレームリクエスト: `/frames/1`
 - マルチフレームリクエスト: `/frames/1,2,3,4`
4. `datastoreId`、`studyInstanceUID`、およびの値を使用して `imageSetId`、リクエストの URL `seriesInstanceUID` `sopInstanceUID` を作成します `frameList`。次の例の URL パス全体を表示するには、コピーボタンにスクロールします。URL は 形式です。

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/studies/study-instance-uid/series/series-instance-uid/instances/sop-instance-uid/frames/1?imageSetId=image-set-id
```

5. リクエストを準備して送信します。は署名 [AWS バージョン 4](#) の署名プロトコルで HTTP GET リクエスト `GetDICOMInstanceFrames` 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、HealthImaging からの `multipart` レスポンスでイメージフレームを取得します。

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/frames/1?
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: multipart/related; type=application/octet-stream; transfer-
syntax=1.2.840.10008.1.2.1'
```

Note

transfer-syntax UID はオプションであり、含まれていない場合はデフォルトで Explicit VR Little Endian になります。ELE へのトランスコードが不可能な場合 (警告付きのインポートのため)、ピクセルはトランスコードなしで返されます。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 (可逆画像フレームのデフォルト)
- transfer-syntax=* その場合、イメージフレーム (複数可) は保存された転送構文で返されます。
- 高スループット JPEG 2000 with RPCL Options Image Compression (Lossless Only) - 1.2.840.10008.1.2.4.202 - インスタンスが HealthImaging に次のように保存されている場合 1.2.840.10008.1.2.4.202
- JPEG 2000 可逆 - 1.2.840.10008.1.2.4.90 - インスタンスが可逆として HealthImaging に保存されている場合。
- JPEG ベースライン (プロセス 1): 損失 JPEG 8 ビットイメージ圧縮のデフォルト転送構文 - 1.2.840.10008.1.2.4.50 インスタンスが HealthImaging にとして保存されている場合 1.2.840.10008.1.2.4.50
- JPEG 2000 Image Compression - 1.2.840.10008.1.2.4.91 - インスタンスが HealthImaging に次のように保存されている場合 1.2.840.10008.1.2.4.91
- 高スループット JPEG 2000 イメージ圧縮 - 1.2.840.10008.1.2.4.203 - インスタンスが HealthImaging にとして保存されている場合 1.2.840.10008.1.2.4.203

- JPEG XL イメージ圧縮 - 1.2.840.10008.1.2.4.112 - インスタンスが HealthImaging に次のように保存されている場合 1.2.840.10008.1.2.4.112
- HealthImaging に保存されているインスタンスは、[転送構文](#)の MPEG ファミリー (MPEG2, MPEG-4 AVC/H.264、HEVC/H.265 を含む) でエンコードされた 1 つ以上のイメージフレームを使用して、対応する転送構文 UID で取得できます。たとえば、インスタンスが MPEG2 Main Profile Main Level として保存1.2.840.10008.1.2.4.100されている場合です。
- リクエストされた転送構文NotAcceptableExceptionをストアド転送構文に基づいて返すことができない場合、またはインスタンスに特定の処理警告がある場合、406 が表示されることがあります。この場合、 で呼び出しを再試行しますtransfer-syntax=*。

詳細については、「[サポートされる転送構文](#)」および「[AWS HealthImaging の画像フレームデコードライブラリ](#)」を参照してください。

HealthImaging からの DICOM バルクデータの取得

GetDICOMBulkdata アクションを使用して、HealthImaging データストアの DICOM メタデータから分離されたバイナリデータを取得します。インスタンスまたはシリーズのメタデータを取得する場合、1MB を超えるバイナリ属性はインライン値BulkDataURIではなく で表されます。HealthImaging データストア内の任意のプライマリイメージセットのバイナリデータを取得するには、メタデータレスポンスでBulkDataURI提供される を使用します。非プライマリイメージセットのバルクデータを取得するには、イメージセット ID をクエリパラメータとして指定します。

DICOM バルクデータを取得するには

GetDICOMInstanceMetadata や などの HealthImaging DICOMweb WADO-RS アクションから DICOM メタデータを取得すると、次に示すようにGetDICOMSeriesMetadata、大きなバイナリ属性が BulkDataURIsにインラインで置き換えられます。

```
"00451026": {
  "vr": "UN",
  "BulkDataURI": "https://dicom-medical-imaging.us-west-2.amazonaws.com/datastore/
<datastoreId>/studies/<StudyInstanceUID>/series/<SeriesInstanceUID>/instances/
<SOPInstanceUID>/bulkdata/<bulkdataUriHash>"
}
```

GetDICOMBulkdata アクションを使用して DICOM 要素を取得するには、次のステップを使用します。

1. フォームの の値を使用してBulkDataURI、リクエストの URL を作成します。

```
https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/  
studies/study-instance-uid/series/series-instance-uid/instances/sop-instance-uid/  
bulkdata/bulkdata-uri-hash
```

2. [AWS 署名バージョン 4](#) の署名プロトコルを使用して、GetDICOMBulkdata コマンドを HTTP GET リクエストとして発行します。次のコード例では、curl コマンドラインツールを使用して、プライマリイメージセットから DICOM 要素を取得します。

```
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/1.2.840.10008.5.1.4.1.1.7/bulkdata/b026324c6904b2a9cb4b88d6d61c81d1' \  
  --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/octet-stream' \  
  --output 'bulkdata.bin'
```

非プライマリイメージセットから DICOM データ要素を取得するには、ImageSetId パラメータを指定します。

```
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/1.2.840.10008.5.1.4.1.1.7/bulkdata/b026324c6904b2a9cb4b88d6d61c81d1?  
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/octet-stream' \  
  --output 'bulkdata.bin'
```

Note

imageSetId パラメータは、プライマリ以外のイメージセットのバルクデータを取得するために必要です。GetDICOMBulkdata アクションは、datastoreId、studyInstanceUID、seriesInstanceUIDおよびSOPInstanceUIDが指定されている場合にのみ、プライマリイメージセットのバルクデータを返します (なし imageSetID)。

HealthImaging での DICOM データの検索

AWS HealthImaging は、[DICOMweb の "DO-RS APIs](#) の表現を提供し、患者 ID で検査、シリーズ、インスタンスを検索し、さらに使用するための一意の識別子を受け取ります。HealthImaging の DICOMweb "DO-RS APIs は、HealthImaging に保存されているデータを検索する方法に柔軟性を提供し、レガシーアプリケーションとの相互運用性を提供します。

[重要]

HealthImaging の DICOMweb APIs を使用して、画像セット情報を "DO-RS で返すことができます。HealthImaging DICOMweb APIs、特に明記されていない限り、[画像セット](#)のみを参照します。HealthImaging [クラウドネイティブアクション](#)、または DICOMweb アクションのオプションの画像セットパラメータを使用して、プライマリ以外の画像セットを取得します。HealthImaging の DICOMweb APIs を使用して、DICOMweb 準拠のレスポンスで画像セット情報を返すことができます。

HealthImaging DICOMweb の "DO-RS アクションは、最大 10,000 件のレコードを返すことができます。10,000 を超えるリソースが存在する場合、これらは "DO-RS アクションでは取得できませんが、DICOMweb WADO-RS アクションまたは [クラウドネイティブアクション](#) で取得できます。

このセクションに記載されている APIs は、ウェブベースの医療画像用の DICOMweb ("DO-RS) 標準に準拠して構築されています。AWS CLI および AWS SDKs では提供されません。

HealthImaging の DICOMweb 検索 APIs

次の表は、HealthImaging でデータを検索するために使用できる DICOMweb "DO-RS APIs のすべての HealthImaging 表現を示しています。

HealthImaging DICOMweb の "DO-RS APIs の表現

名前	説明
SearchDICOMStudies	GET リクエストを使用して検索クエリ要素を指定して、HealthImaging で DICOM 検査を検索します。調査結果は JSON 形式で返され、最終更新日、日付の降順 (最新から最新) 順に並べられます。「 検査を検索する 」を参照してください。
SearchDICOMSeries	GET リクエストを使用して検索クエリ要素を指定して、HealthImaging で DICOM シリーズを検索します。系列の検索結果は JSON 形式で返され、昇順 (最も古いものから最新のもの) Series Number (0020, 0011) に並べられます。「 シリーズの検索 」を参照してください。
SearchDICOMInstances	GET リクエストを使用して検索クエリ要素を指定して、HealthImaging で DICOM インスタンスを検索します。インスタンスの検索結果は JSON 形式で返され、昇順 (最も古いものから最新のもの) Instance Number (0020, 0013) に並べられます。「 インスタンスの検索 」を参照してください。

HealthImaging でサポートされている DICOMweb クエリタイプ

HealthImaging は、治験、シリーズ、SOP インスタンスレベルで "DO-RS 階層型リソースクエリをサポートしています。HealthImaging に "DO-RS 階層検索を使用する場合:

- スタディを検索すると、スタディのリストが返されます。
- 研究のシリーズを検索するには既知の `StudyInstanceUID` が、シリーズのリストが返されます。
- インスタンスのリストを検索するには、既知の `StudyInstanceUID` と `SeriesInstanceUID` が必要です。

次の表は、HealthImaging でデータを検索するためにサポートされている "DO-RS 階層クエリタイプ" を示しています。

HealthImaging でサポートされている "DO-RS クエリタイプ"

クエリタイプ	例
属性値のクエリ	<p>で、スタディ内のすべてのシリーズを検索し ずmodality=CT 。</p> <p>.../studies/1.3.6.1.4.1.145 19.5.2.1.6279.6001.10137060 5276577556143013894866/series? 00080060=CT</p> <p>患者 ID と検査日がそれぞれこれらの値である すべての検査を検索します。</p> <p>.../studies?PatientID=1123581 3&StudyDate=20130509</p>
キーワードクエリ	<p>SeriesInstanceUID キーワードを使用して すべてのシリーズを検索します。</p> <p>.../studies/1.3.6.1.4.1.145 19.5.2.1.6279.6001.10137060 5276577556143013894866/seri es?SeriesInstanceUID=1.3.6. 1.4.1.14519.5.2.1.6279.6001 .101370605276577556143013894868</p>
クエリにタグを付ける	<p>グループ/要素フォームで渡されたクエリパラ メータを使用してタグを検索します。</p> <p>0020000D のような {group}{element}</p>
範囲クエリ	<p>...?Modality=CT&StudyDate=A ABYYYY-BBCCYYYY</p>

クエリタイプ	例
limit および offset を使用した結果のページング	<pre>.../studies?limit=1&offset=0&00080020=20000101</pre> <p>制限パラメータとオフセットパラメータを使用して、検索レスポンスをページ分割できます。制限のデフォルト値は 1000 です。最大値 AWS HealthImaging エンドポイントとクォータ については、「」を参照してください。</p> <p>最大制限 = 1000、最大オフセット = 9000</p>

クエリタイプ	例
ワイルドカードクエリ	<p>ワイルドカードクエリは、「*」と「?」を使用した検索の柔軟性を高めます。「*」は任意の文字シーケンス (長さがゼロの値を含む) に一致し、「?」は任意の 1 文字に一致します。</p> <p>StudyDescription に「Nuclear」が含まれているデータストア内のすべての研究を検索します。</p> <pre>.../studies?StudyDescription=*Nuclear*</pre> <p>StudyDescription が「Nuclear」で終わるすべての研究を検索します。</p> <pre>.../studies?StudyDescription=Nuclear</pre> <p>StudyDescription が「Nuclear」で始まるすべての研究を検索します。</p> <pre>.../studies?StudyDescription=Nuclear*</pre> <p>PatientID が 200965981 の後に 3 文字のみを持つすべての治験を検索します。</p> <pre>.../studies?PatientID=200965981???</pre>

クエリタイプ	例
FuzzyMatching クエリ	<p>ファジーマッチングオプションのクエリパラメータを追加して、名前の DICOM 属性 (PatientName (0010,0010), ReferringPhysicianName(0008,0090)) であいまいマッチングを有効にします。</p> <pre>.../studies?fuzzymatching=true&PatientName="Thomas^Albert"</pre> <p>このクエリは、PatientName 値の任意の部分で大文字と小文字を区別しないプレフィックス単語マッチングを実行します。「thomas」、「Albert」、「Thomas Albert」、「Thomas^Albert」などの PatientName 値の結果を返しますが、「hom」や「ber」は返しません。</p>

トピック

- [HealthImaging での DICOM 検査の検索](#)
- [HealthImaging での DICOM シリーズの検索](#)
- [HealthImaging での DICOM インスタンスの検索](#)

HealthImaging での DICOM 検査の検索

SearchDICOMStudies API を使用して、HealthImaging [データストア](#)内の DICOM 検査を検索します。HealthImaging で DICOM スタディを検索するには、サポートされている DICOM データ要素 (属性) を含む URL を作成します。調査結果は JSON 形式で返され、最終更新日、日付の降順 (最新から最新) 順に並べられます。

DICOM 検査を検索するには

1. HealthImaging regionとdatastoreId値を収集します。詳細については、「[データストアのブローパティの取得](#)」を参照してください。
2. 該当するすべての治験要素を含む、リクエストの URL を作成します。次の例の URL パス全体を表示するには、コピーボタンをスクロールします。URL は 形式です。

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastoreId/
studies[?query]
```

の学習要素 SearchDICOMStudies

DICOM 要素タグ	DICOM 要素名
(0008,0020)	Study Date
(0008,0030)	StudyTime
(0008,0050)	Accession Number
(0008,0061)	Modalities in Study
(0008,0090)	Referring Physician Name
(0008,1030)	Study Description
(0010,0010)	Patient Name
(0010,0020)	Patient ID
(0010,0030)	Patient BirthDate
(0010,0032)	Patient BirthTime
(0020,000D)	Study Instance UID
(0020,0010)	Study ID

- リクエストを準備して送信します。は署名[AWS バージョン 4](#)の署名プロトコルで HTTP GET リクエスト SearchDICOMStudies を使用します。次の例では、curl コマンドラインツールを使用して DICOM 検査に関する情報を検索します。

curl

```
curl --request GET \
  "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
  studies[?query]"
  --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+json' \
--output results.json
```

調査結果は JSON 形式で返され、最終更新日、日付の降順 (最新から最新) 順に並べられます。

HealthImaging での DICOM シリーズの検索

SearchDICOMSeries API を使用して、HealthImaging [データストア](#)で DICOM シリーズを検索します。HealthImaging で DICOM シリーズを検索するには、サポートされている DICOM データ要素 (属性) を含む URL を作成します。系列の検索結果は、昇順 (最新から最新) の JSON 形式で返されます。

DICOM シリーズを検索するには

1. HealthImaging regionとdatastoreId値を収集します。詳細については、「[データストアのロパティの取得](#)」を参照してください。
2. StudyInstanceUID 値を収集します。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. 該当するすべてのシリーズ要素を含む、リクエストの URL を作成します。次の例の URL パス全体を表示するには、コピーボタンをスクロールします。URL は 形式です。

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastoreId/studies/StudyInstanceUID/series[?query]
```

のシリーズ要素 SearchDICOMSeries

DICOM 要素タグ	DICOM 要素名
(0008,0060)	Modality
(0020,000E)	Series Instance UID

4. リクエストを準備して送信します。は署名[AWS バージョン 4](#)の署名プロトコルで HTTP GET リクエストSearchDICOMSeriesを使用します。次の例では、curlコマンドラインツールを使用して DICOM シリーズ情報を検索します。

curl

```
curl --request GET \  
  "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/  
studies/StudyInstanceUID/series[?query]" \  
  --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+json' \  
  --output results.json
```

系列の検索結果は、JSON 形式で、昇順 (最も古いものから最新のもの) Series Number (0020,0011) の順に返されます。

HealthImaging での DICOM インスタンスの検索

SearchDICOMInstances API を使用して、HealthImaging [データストア](#)内の DICOM インスタンスを検索します。HealthImaging で DICOM インスタンスを検索するには、サポートされている DICOM データ要素 (属性) を含む URL を作成します。インスタンスの結果は、昇順 (最新から最新) の JSON 形式で返されます。

DICOM インスタンスを検索するには

1. HealthImaging regionとdatastoreId値を収集します。詳細については、「[データストアのプロパティの取得](#)」を参照してください。
2. StudyInstanceUID および の値を収集しますSeriesInstanceUID。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. 該当するすべての検索要素を含む、リクエストの URL を作成します。次の例の URL パス全体を表示するには、コピーボタンをスクロールします。URL は 形式です。

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastoreId/  
studies/StudyInstanceUID/series/SeriesInstanceUID/instances[?query]
```

のインスタンス要素 SearchDICOMInstances

DICOM 要素タグ	DICOM 要素名
(0008,0016)	SOP Class UID
(0008,0018)	SOP Instance UID
(0008,1196)	WarningReason

HealthImaging は DICOM 要素 [\(0008,1196\)](#) を使用してインポート警告コードを保持します。インポート警告コードはインスタンスレベルで検索できます。インポート警告コードは、ワイルドカードまたは特定の警告コードで検索できます。「[HealthImaging 警告コード](#)」を参照してください。

- リクエストを準備して送信します。は署名 [AWS バージョン 4](#) の署名プロトコルで HTTP GET リクエスト SearchDICOMInstances を使用します。次の例では、curl コマンドラインツールを使用して DICOM インスタンスに関する情報を検索します。

curl

```
curl --request GET \
  "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
  studies/StudyInstanceUID/series/SeriesInstanceUID/instances[?query]"
  --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+json' \
  --output results.json
```

インスタンスの検索結果は、JSON 形式で、昇順 (最も古いものから最新のもの) Instance Number (0020,0013) の順に返されます。

DICOMweb APIs の OIDC 認証

AWS HealthImaging は、既存の Signature Version 4 ([SigV4](#)) 認証に加えて、[OpenID Connect \(OIDC\)](#) を使用した DICOMweb API リクエストの [OAuth 2.0](#) ベースの認証をサポートしています。[AWS SigV4](#) OIDC を使用すると、HealthImaging を外部 ID プロバイダー (IdPs) と直接統合し、標準

ベースのアプリケーションに HealthImaging DICOMweb エンドポイントを介して医療画像データへのアクセスを提供できます。各アプリケーションには AWS 認証情報は必要ありません。

トピック

- [Lambda オーソライザーによるカスタムトークン検証](#)
- [OIDC 認証用の AWS Lambda オーソライザーを設定する](#)

Lambda オーソライザーによるカスタムトークン検証

HealthImaging は、Lambda オーソライザーを使用するアーキテクチャを通じて OIDC サポートを実装し、お客様が独自のトークン検証ロジックを実装できるようにします。この設計により、OIDC 互換 ID プロバイダー (IdPs) の多様な環境とさまざまなトークン検証方法に対応して、トークンの検証方法とアクセス決定の適用方法を柔軟に制御できます。

認証のフロー

認証の仕組みの概要は次のとおりです。

1. クライアントが DICOMweb API を呼び出す: アプリケーションは選択した OIDC ID プロバイダーで認証し、署名付き ID トークン (JWT) を受け取ります。DICOMweb HTTP リクエストごとに、クライアントは認可ヘッダー (通常はベアラートークン) に OIDC アクセストークンを含める必要があります。リクエストがデータに到達する前に、HealthImaging は受信リクエストからこのトークンを抽出し、設定した Lambda オーソライザーを呼び出します。
 - a. ヘッダーは通常、 の形式に従います `Authorization: Bearer <token>`。
2. 初期検証: HealthImaging は、Lambda 関数を不必要に呼び出すことなく、明らかに無効または期限切れのトークンをすばやく拒否するために、アクセストークンクレームを検証します。HealthImaging は、Lambda オーソライザーを呼び出す前に、アクセストークン内の特定の標準クレームの初期検証を実行します。
 - a. `iat` (発行時): HealthImaging は、トークンの発行時間が許容範囲内であるかどうかをチェックします。
 - b. `exp` (有効期限): HealthImaging はトークンの有効期限が切れていないことを確認します。
 - c. `nbf` (時間前ではない): 存在する場合、HealthImaging は有効な開始時刻より前にトークンが使用されていないことを確認します。
3. HealthImaging は Lambda オーソライザーを呼び出します。最初のクレーム検証に合格すると、HealthImaging は追加のトークン検証をお客様が設定した Lambda オーソライザー関数に

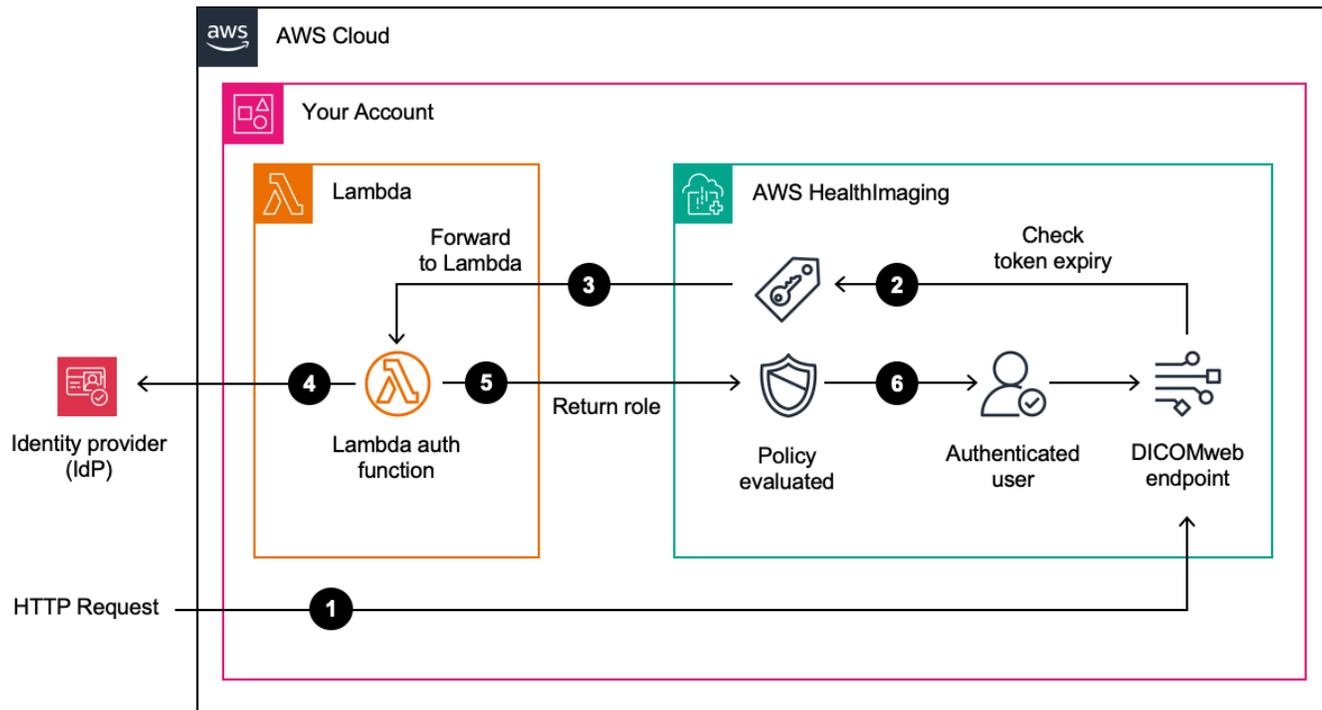
委任します。HealthImaging は、抽出されたトークンおよびその他の関連するリクエスト情報を Lambda 関数に渡します。Lambda 関数は、トークンの署名とクレームを検証します。

4. ID プロバイダーとの検証: Lambda には、ID トークンの署名をチェックし、より広範なトークン検証 (発行者、対象者、カスタムクレームなど) を実行し、必要に応じてそれらのクレームを IdP に対して検証するカスタムコードが含まれています。
5. オーソライザーはアクセスポリシーを返します。検証に成功すると、Lambda 関数は認証された使用に対する適切なアクセス許可を決定します。次に、Lambda オーソライザーは、付与される一連のアクセス許可を表す IAM ロールの amazon リソースネーム (ARN) を返します。
6. リクエストの実行: 引き受けた IAM ロールに必要なアクセス許可がある場合、HealthImaging はリクエストされた DICOMWeb リソースの返却に進みます。アクセス許可が不十分な場合、HealthImaging はリクエストを拒否し、適切なエラーレスポンスエラー (403 Forbidden) を返します。

Note

AWS HealthImaging サービスによって管理されていないオーソライザー Lambda 関数。AWS アカウントで実行されます。お客様は、関数の呼び出しと実行時間に対して、HealthImaging 料金とは別に課金されます。

アーキテクチャの概要



Lambda オーソライザーを使用した OIDC 認証ワークフロー

前提条件

アクセストークンの要件

HealthImaging では、アクセストークンが JSON Web Token (JWT) 形式である必要があります。多くの ID プロバイダー (IDPs) ではアクセストークンフォームを選択または設定できます。統合を進める前に、選択した IDP が JWT トークンを発行できることを確認します。

トークンの形式

アクセストークンは JWT (JSON Web Token) 形式である必要があります

必要な請求

exp (有効期限)

トークンがいつ無効になるかを指定する必須クレーム。

- UTC の現在の時刻より後である必要があります
- トークンが無効になったときを表します

iat (発行先)

トークンの発行日時を指定する必須クレーム。

- UTC の現在の時刻より前である必要があります
- UTC の現在の時刻より 12 時間前にすることはできません
- これにより、トークンの最大有効期間は 12 時間になります。

nbf (以前ではありません)

トークンを使用できる最も早い時刻を指定するオプションクレーム。

- 存在する場合、HealthImaging によって評価されます
- トークンを受け入れない時間を指定します。

Lambda オーソライザーの応答時間の要件

HealthImaging は、最適な API パフォーマンスを確保するために、Lambda オーソライザーレスポンスに厳格なタイミング要件を適用します。Lambda 関数は 1 秒以内に を返す必要があります。

ベストプラクティス

トークン検証の最適化

- 可能であれば JWKS (JSON ウェブキーセット) をキャッシュする
- 可能な場合は有効なアクセストークンをキャッシュする
- ID プロバイダーへのネットワーク呼び出しを最小限に抑える
- 効率的なトークン検証ロジックを実装する

Lambda 設定

- Python および Node.js ベースの関数は通常、より高速に初期化します。
- ロードする外部ライブラリの量を減らす
- 一貫したパフォーマンスを確保するために適切なメモリ割り当てを設定する
- CloudWatch メトリクスを使用して実行時間をモニタリングする

OIDC 認証の有効化

- OIDC 認証は、新しいデータストアを作成する場合にのみ有効にできます
- 既存のデータストアの OIDC の有効化は API ではサポートされていません
- 既存のデータストアで OIDC を有効にするには、サポートに連絡する必要があります AWS。

OIDC 認証用の AWS Lambda オーソライザーを設定する

このガイドでは、選択した ID プロバイダー (IdP) が HealthImaging OIDC 認証機能の要件と互換性のあるアクセストークンを提供するように設定されていることを前提としています。

1. DICOMWeb API アクセス用の IAM ロールを設定する

Lambda オーソライザーを設定する前に、DICOMWeb API リクエストを処理するときに HealthImaging が引き受ける IAM ロールを作成します。オーソライザー Lambda 関数は、トークンの検証に成功した後、これらのロール ARN のいずれかを返します。これにより、HealthImaging は適切なアクセス許可でリクエストを実行できます。

1. 必要な DICOMWeb API 権限を定義する IAM ポリシーを作成します。利用可能なアクセス許可については、HealthImaging ドキュメントの[DICOMweb の使用](#) セクションを参照してください。
2. 以下の IAM ロールを作成します。
 - これらのポリシーをアタッチする
 - AWS HealthImaging サービスプリンシパル (`medical-imaging.amazonaws.com`) がこれらのロールを引き受けることを許可する信頼関係を含めます。

以下は、関連付けられたロールが HealthImaging DICOMWeb 読み取り専用 API にアクセスすることを許可するポリシーの例です。

ロール (複数可) に関連付ける必要がある信頼関係ポリシーの例を次に示します。

JSON

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

```
        "Effect": "Allow",
        "Principal": {
            "Service": "medical-imaging.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
]
}
```

次のステップで作成する Lambda オーソライザーは、トークンクレームを評価し、適切なロールの ARN を返すことができます。次に、AWS HealthImaging はこのロールを偽装して、対応するアクセス許可で DICOMWeb API リクエストを実行します。

例えば、次のようになります。

- 「管理者」クレームを持つトークンは、フルアクセスを持つロールの ARN を返す場合があります
- 「リーダー」クレームを持つトークンは、読み取り専用アクセスを持つロールの ARN を返す場合があります
- 「department_A」クレームを持つトークンは、その部門のアクセスレベルに固有のロールの ARN を返す場合があります

このメカニズムにより、IAM ロールを介して IdP の認可モデルを特定の AWS HealthImaging アクセス許可にマッピングできます。

2. Lambda オーソライザー関数の作成と設定

JWT トークンを検証し、トークンクレーム評価に基づいて適切な IAM ロール ARN を返す Lambda 関数を作成します。この関数はヘルスイメージングサービスによって呼び出され、HealthImaging データストア ID、DICOMWeb オペレーション、HTTP リクエストで見つかったアクセストークンを含むイベントを渡します。

```
{
  "datastoreId": "{datastore id}",
  "operation": "{Healthimaging API name e.g. GetDICOMInstance}",
  "bearerToken": "{access token}"
}
```

Lambda オーソライザー関数は、次の構造を持つ JSON レスポンスを返す必要があります。

```
{
  "isTokenValid": {true or false},
  "roleArn": "{role arn or empty string meaning to deny the request explicitly}"
}
```

詳細については、実装例を参照してください。

Note

DICOMWeb リクエストは、Lambda オーソライザーによってアクセストークンが検証された後にのみ応答されるため、この関数の実行を可能な限り速くして、最適な DICOMWeb API 応答時間を提供することが重要です。

HealthImaging サービスが Lambda オーソライザー関数を呼び出す権限を付与されるようにするには、HealthImaging サービスが関数を呼び出すことを許可するリソースポリシーが必要です。このリソースポリシーは、Lambda 設定タブのアクセス許可メニューまたは以下を使用して作成できます AWS CLI。

```
aws lambda add-permission \
  --function-name YourAuthorizerFunctionName \
  --statement-id HealthImagingInvoke \
  --action lambda:InvokeFunction \
  --principal medical-imaging.amazonaws.com
```

このリソースポリシーにより、HealthImaging サービスは DICOMWeb API リクエストを認証するときに Lambda オーソライザーを呼び出すことができます。

Note

Lambda リソースポリシーは、特定の HealthImaging データストアの ARN に一致する「ArnLike」条件で後で更新できます。

Lambda リソースポリシーの例を次に示します。

3. OIDC 認証を使用して新しいデータストアを作成する

OIDC 認証を有効にするには、`lambda-authorizer-arn` パラメータ AWS CLI を持つ を使用して新しいデータストアを作成する必要があります。サポートに連絡して AWS なければ、既存のデータストアで OIDC 認証を有効にすることはできません。

OIDC 認証を有効にして新しいデータストアを作成する方法の例を次に示します。

```
aws medical-imaging create-datastore \  
  --datastore-name YourDatastoreName \  
  --lambda-authorizer-arn YourAuthorizerFunctionArn
```

`get-datastore` コマンドを使用して AWS CLI、特定のデータストアで OIDC 認証機能が有効になっているかどうか、および属性 `lambdaAuthorizerArn` が存在するかどうかを確認できます。

```
aws medical-imaging get-datastore --datastore-id YourDatastoreId
```

```
{  
  "datastoreProperties": {  
    "datastoreId": YourdatastoreId,  
    "datastoreName": YourDatastoreName,  
    "datastoreStatus": "ACTIVE",  
    "lambdaAuthorizerArn": YourAuthorizerFunctionArn,  
    "datastoreArn": YourDatastoreArn,  
    "createdAt": "2025-09-30T14:16:04.015000-05:00",  
    "updatedAt": "2025-09-30T14:16:04.015000-05:00"  
  }  
}
```

Note

AWS CLI データストア作成コマンドの実行ロールには、Lambda オーソライザー関数を呼び出すための適切なアクセス許可が必要です。これにより、悪意のあるユーザーがデータストアオーソライザー設定を通じて不正な Lambda 関数を実行できる特権エスケーレーション攻撃が軽減されます。

例外コード

認証に失敗した場合、HealthImaging は次の HTTP エラーレスポンスコードと本文メッセージを返します。

Condition	AHI レスポンス
Lambda オーソライザーが存在しないか、無効です	424 オーソライザーの設定ミス
実行失敗によりオーソライザーが終了しました	424 オーソライザーが失敗しました
マッピングされていない他のオーソライザーエラー	424 オーソライザーが失敗しました
オーソライザーが無効な/不正な形式のレスポンスを返しました	424 オーソライザーの設定ミス
オーソライザーが 1 秒以上実行されました	408 オーソライザータイムアウト
トークンの有効期限が切れているか、無効です	403 無効または期限切れのトークン
オーソライザーの設定ミスにより、AHI は返された IAM ロールをフェデレーションできません	424 オーソライザーの設定ミス
オーソライザーが空のロールを返しました	403 アクセスが拒否されました
返されたロールは呼び出せません (assume-role/trust misconfig)	424 オーソライザーの設定ミス

Condition	AHI レスポンス
リクエストレートが DICOMweb Gateway の制限を超えています	429 リクエストが多すぎる
データストア、リターンロール、またはオーソライザーのクロスアカウント/クロスリージョン	424 オーソライザークロスアカウント/クロスリージョンアクセス

実装例

この Python の例は、HealthImaging イベントからの AWS Cognito アクセストークンを検証し、適切な DICOMWeb 権限を持つ IAM ロール ARN を返す Lambda オーソライザー関数を示しています。

Lambda オーソライザーは、2 つのキャッシュメカニズムを実装して、外部呼び出しと応答のレイテンシーを削減します。JWKS (JSON ウェブキーセット) は 1 時間に 1 回フェッチされ、関数の一時フォルダに保存されるため、後続の関数呼び出しはパブリックネットワークからフェッチする代わりにローカルで読み取ることができます。token_cache ディクショナリオブジェクトがこの Lambda 関数のグローバルコンテキストでインスタンス化されていることもわかります。グローバル変数は、同じウォームされた Lambda コンテキストを再利用するすべての呼び出しによって共有されます。これにより、正常に検証されたトークンをこのディクショナリに保存し、この同じ Lambda 関数の次の実行中にすばやく検索できます。キャッシュ方法は、ほとんどの ID プロバイダーから発行されたアクセストークンに適合する可能性がある一般論的なアプローチを表します。AWS Cognito 固有のキャッシュオプションについては、[AWS Cognito ドキュメント](#)の「[ユーザープールの管理](#)」セクションと「[キャッシュ](#)」セクションを参照してください。

```
import json
import os
import time
import logging
from jose import jwk, jwt
from jose.exceptions import ExpiredSignatureError, JWTClaimsError, JWTError
import requests
import tempfile

# Configure logging
logger = logging.getLogger()
```

```
log_level = os.environ.get('LOG_LEVEL', 'WARNING').upper()
logger.setLevel(getattr(logging, log_level, logging.WARNING))

# Global token cache with TTL
token_cache = {}

# JWKS cache file path
JWKS_CACHE_FILE = os.path.join(tempfile.gettempdir(), 'jwks.json')
JWKS_CACHE_TTL = 3600 # 1 hour

# Load environment variables once
USER_POOL_ID = os.environ['USER_POOL_ID']
CLIENT_ID = os.environ['CLIENT_ID']
ROLE_ARN = os.environ.get('AHIDICOMWEB_READONLY_ROLE_ARN', '')

def cleanup_expired_tokens():
    """Remove expired tokens from cache"""
    now = int(time.time())
    expired_keys = [token for token, data in token_cache.items() if now >
data['cache_expiry']]
    for token in expired_keys:
        del token_cache[token]

def get_cached_jwks():
    """Get JWKS from cache file if valid, otherwise return None """
    try:
        if os.path.exists(JWKS_CACHE_FILE):
            # Check if cache file is still valid
            cache_age = time.time() - os.path.getmtime(JWKS_CACHE_FILE)
            if cache_age < JWKS_CACHE_TTL:
                with open(JWKS_CACHE_FILE, 'r') as f:
                    jwks = json.load(f)
                    logger.debug(f'Using cached JWKS (age: {int(cache_age)}s)')
                    return jwks
            else:
                logger.debug(f'JWKS cache expired (age: {int(cache_age)}s)')
    except Exception as e:
        logger.debug(f'Error reading JWKS cache: {e}')

    return None

def cache_jwks(jwks):
    """Cache JWKS to file"""
    try:
```

```
        with open(JWKS_CACHE_FILE, 'w') as f:
            json.dump(jwks, f)
            logger.debug('JWKS cached successfully')
    except Exception as e:
        logger.debug(f'Error caching JWKS: {e}')

def fetch_jwks(jwks_url):
    """Fetch JWKS from URL and cache it"""
    logger.debug('Fetching JWKS from URL')
    jwks = requests.get(jwks_url, timeout=10).json()
    # Convert to dict for faster lookups
    jwks['keys_by_kid'] = {key['kid']: key for key in jwks['keys']}
    cache_jwks(jwks)
    return jwks

def is_token_cached(token):
    if token not in token_cache:
        return None

    cached = token_cache[token]
    now = int(time.time())

    if now > cached['cache_expiry']:
        del token_cache[token]
        return None

    return cached

def cache_token(token, payload):
    now = int(time.time())
    token_exp = payload.get('exp')
    cache_expiry = min(now + 60, token_exp) # 1 minute or token expiry, whichever is
    sooner

    token_cache[token] = {
        'payload': payload,
        'cache_expiry': cache_expiry,
        'role_arn': ROLE_ARN
    }

def handler(event, context):
    cleanup_expired_tokens() # start be removing expired tokens from the cache
    try:
        # Extract token from bearerToken or authorizationToken field
```

```
token = event.get('bearerToken')
if not token:
    raise Exception('No token provided')

# Check cache first
cached = is_token_cached(token)
if cached:
    logger.debug('Token found in cache, skipping verification')
    return {
        'isTokenValid': True,
        'roleArn': cached['role_arn']
    }

# Get Cognito configuration
region = context.invoked_function_arn.split(':')[3]

# Get JWKS (cached or fresh)
jwks_url = f'https://cognito-idp.{region}.amazonaws.com/{USER_POOL_ID}/.well-known/jwks.json'
jwks = get_cached_jwks()
if not jwks:
    jwks = fetch_jwks(jwks_url)

# Decode token header to get kid
headers = jwt.get_unverified_headers(token)
kid = headers['kid']

# Find the correct key
key = None
for jwk_key in jwks['keys']:
    if jwk_key['kid'] == kid:
        key = jwk_key
        break

if not key:
    # Key not found - try refreshing JWKS in case of key rotation
    logger.debug('Key not found in cached JWKS, fetching fresh JWKS')
    jwks = fetch_jwks(jwks_url)
    for jwk_key in jwks['keys']:
        if jwk_key['kid'] == kid:
            key = jwk_key
            break

if not key:
```

```
        raise Exception('Public key not found')

    # Construct the public key
    public_key = jwk.construct(key)

    # Verify and decode the token (includes expiry validation)
    payload = jwt.decode(
        token,
        public_key,
        algorithms=['RS256'],
        audience=CLIENT_ID,
        issuer=f'https://cognito-idp.{region}.amazonaws.com/{USER_POOL_ID}'
    )

    logger.debug('Token validated successfully')
    logger.debug('User: %s', payload.get('username', 'unknown'))

    # Cache the validated token
    cache_token(token, payload)

    # Return authorization response
    return {
        'isTokenValid': True,
        'roleArn': ROLE_ARN
    }

except ExpiredSignatureError:
    logger.debug('Token expired')
    return {
        'isTokenValid': False,
        'roleArn': ''
    }

except JWTClaimsError:
    logger.debug('Invalid token claims')
    return {
        'isTokenValid': False,
        'roleArn': ''
    }

except JWTError as e:
    logger.debug('JWT validation error: %s', e)
    return {
        'isTokenValid': False,
        'roleArn': ''
    }
```

```
except Exception as e:
    logger.debug('Authorization failed: %s', e)
    return {
        'isTokenValid': False,
        'roleArn': ''
    }
```

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[画像フレーム](#) (ピクセルデータ) で構成される[画像セット](#)に変換します。

[重要]

HealthImaging インポートジョブは、DICOM インスタンスバイナリ (.dcm ファイル) を処理し、画像セットに変換します。HealthImaging [クラウドネイティブアクション](#) (APIs) を使用して、データストアとイメージセットを管理します。HealthImaging の [DICOMweb サービス表現](#)を使用して DICOMweb レスポンスを返します。

以下のトピックでは、AWS マネジメントコンソール、AWS CLI、AWS SDKs 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 HealthImaging で[データストア](#)を作成したら、Amazon S3 入力バケットから医療画像データをデータストアにインポートして[画像セット](#)を作成する必要があります。AWS マネジメントコンソール、AWS CLI、および 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。

DICOM P10 データを AWS HealthImaging データストアにインポートすると、サービスは[メタデータ要素](#)に基づいて、治験 UID、シリーズ UID、インスタンス UID の DICOM 階層に従ってインスタンスを自動的に整理しようとしています。インポートされたデータの[メタデータ要素](#)がデータストア内の既存のプライマリ[イメージセット](#)と競合しない場合、インポートされたデータはプライマリにな

ります。新しくインポートされた DICOM P10 データのメタデータ要素が既存のプライマリ [イメージセット](#) と競合する場合、新しいデータは非プライマリ [イメージセット](#) に追加されます。データインポートが非プライマリ [イメージセット](#) を作成すると、AWS HealthImaging success.ndjson を使用して EventBridge イベントを出力し `isPrimary: False`、に書き込まれたレコードも `importResponse` オブジェクト `isPrimary: False` 内に存在します。

データをインポートすると、HealthImaging は以下を実行します。

- DICOM シリーズで構成されるインスタンスが 1 つのインポートジョブにインポートされ、そのインスタンスがデータストアに既に存在するインスタンスと競合しない場合、すべてのインスタンスが 1 つのプライマリ [イメージセット](#) に整理されます。
- DICOM シリーズで構成されるインスタンスが 2 つ以上のインポートジョブにインポートされ、インスタンスがデータストアに既に存在するインスタンスと競合しない場合、すべてのインスタンスは 1 つのプライマリ [イメージセット](#) として編成されます。
- インスタンスが複数回インポートされた場合、最新バージョンはプライマリ [イメージセット](#) に保存されている古いバージョンを上書きし、プライマリ [イメージセット](#) のバージョン番号が増分されます。

[「イメージセットメタデータの更新」](#) で説明されているステップを使用して、プライマリのインスタンスを更新できます。

インポート中、サイズが 1MB を超えるプライベートタグ (VR タイプ OB、OD、OF、OL、OV、OW、UN) のバイナリ値は、メタデータとは別に保存されます。GetDICOMInstanceMetadata または を使用してこれらのインスタンスのメタデータを取得する場合 GetDICOMSeriesMetadata、これらの大きなバイナリ値は BulkDataURIs に置き換えられ、実際のバイナリデータは GetDICOMBulkdata API を使用して取得できます。

Amazon S3 から 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。

- DICOM シリーズに対応するインスタンスは、プライマリを示す単一の画像セットに自動的に結合されます。
- DICOM P10 データを 1 つのインポートジョブまたは複数のインポートジョブにインポートでき、サービスはインスタンスを DICOM シリーズに対応するプライマリ [イメージセット](#) に整理します。
- 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 のインポートアクションに関連した、エンドポイント、クォータ、スロットリング制限があります。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/",
    "warningsOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/WARNING/",
    "numberOfScannedFiles": 5,
    "numberOfImportedFiles": 3,
    "numberOfFilesWithCustomerError": 2,
    "numberOfFilesWithServerError": 0,
    "numberOfGeneratedImageSets": 2,
```

```

    "imageSetsSummary": [{
      "imageSetId": "12345612345612345678907890789012",
        "numberOfMatchedSOPInstances": 2
      },
      {
        "imageSetId": "12345612345612345678917891789012",
          "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", "isPrimary": True}}
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105630.1.0.1.dcm","importResponse":
{"imageSetId":"12345612345612345678917891789012", "isPrimary": True}}

```

- 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"}}

```

- WARNING フォルダには、正常にインポートされたが警告があるすべての画像 warning.ndjson ファイルの結果を含むファイルが保持されます。次の例は、warning.ndjson ファイルからの出力を示しています。

```

{"inputFile":"dicom_input/warningDicomFile1.dcm","importResponse":
{"imageSetId":"12345612345612345678907890789012","imageSetVersion":1,"isPrimary":true,"warn
[{"warning_reason_code":45330,"type":"InvalidOffsetTable","message":"The file was
imported but contains an invalid offset table, may see issues when retrieving
certain frames."}]}}

```

- インポートジョブはジョブのリストに 90 日間保持され、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 HealthImaging [データストア](#)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

インポートジョブを開始する前に、次の点に注意してください。

- HealthImaging では、転送構文が異なる DICOM P10 ファイルのインポートがサポートされています。一部のファイルはインポート中に元の転送構文エンコードを保持し、他のファイルはデータストアの設定に応じてデフォルトで HTJ2K ロスレスまたは JPEG 2000 ロスレスにトランスコードされます。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。
- 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ています。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 inputOwnerId 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。
- HealthImaging は、インポート中に特定の DICOM 要素に長さの制約を適用します。詳細については、「[DICOM 要素の制約](#)」を参照してください。

以下のメニューでは、 の手順 AWS マネジメントコンソール と、 および SDK の AWS CLI コード例を示します。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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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、AWS SDK for C++ API リファレンスの「[StartDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Starting an import job](#)」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、AWS SDK for Java 2.x API リファレンスの「[StartDICOMImportJob](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',

```

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

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Python (Boto3) API リファレンスの「[StartDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '1234567890123456789012345678901234567890'
```

```
" iv_role_arn = 'arn:aws:iam::123456789012:role/ImportJobRole'
" iv_input_s3_uri = 's3://my-bucket/input/'
" iv_output_s3_uri = 's3://my-bucket/output/'
oo_result = lo_mig->startdicomimportjob(
  iv_jobname = iv_job_name
  iv_datastoreid = iv_datastore_id
  iv_dataaccessrolearn = iv_role_arn
  iv_inputs3uri = iv_input_s3_uri
  iv_outputs3uri = iv_output_s3_uri ).
DATA(lv_job_id) = oo_result->get_jobid( ).
MESSAGE |DICOM import job started with ID: { lv_job_id }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
  MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[StartDICOMImportJob](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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 に書き込まれた出カ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。

以下のメニューでは、 の手順 AWS マネジメントコンソール と、 および SDK の AWS CLI コード例を示します。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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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、AWS SDK for C++ API リファレンスの「[GetDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Getting import job properties](#)」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「[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、AWS SDK for Java 2.x API リファレンスの「[GetDICOMImportJob](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 = "xxxxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a2637936-78ea-44e7-98b8-7a87d95dface',
  //     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/
  output_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxx'/',
  //     submittedAt: 2023-09-19T17:27:25.143Z
  //   }
  // }

  return response;
};
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Python (Boto3) API リファレンスの「[GetDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '12345678901234567890123456789012345678901234567890'  
  " iv_job_id = '12345678901234567890123456789012'  
  oo_result = lo_mig->getdicomimportjob(  
    iv_datastoreid = iv_datastore_id  
    iv_jobid = iv_job_id ).  
  DATA(lo_job_props) = oo_result->get_jobproperties( ).  
  DATA(lv_job_status) = lo_job_props->get_jobstatus( ).  
  MESSAGE |Job status: { lv_job_status }.| TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Job not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[GetDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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 AWS マネジメントコンソールと、AWS CLI および 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ジョブのリストに 90 日間保持され、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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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 Developer Guide」の「[Listing import jobs](#)」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、「AWS SDK for Java 2.x API リファレンス」の「[ListDICOMImportJobs](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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 = "xxxxxxxxxxxxxxxxxxxxxx",
) => {
    const paginatorConfig = {
```

```
    client: medicalImagingClient,
    pageSize: 50,
  };

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

  const 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       endedAt: 2023-09-22T14:49:51.351Z,
  //       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       jobName: 'test-1',
  //       jobStatus: 'COMPLETED',
  //       submittedAt: 2023-09-22T14:48:45.767Z
  //     }
  //   ]
  // }

  return jobSummaries;
};
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  oo_result = lo_mig->listdicomimportjobs( iv_datastoreid =
iv_datastore_id ).
  DATA(lt_jobs) = oo_result->get_jobsummaries( ).
  DATA(lv_count) = lines( lt_jobs ).
  MESSAGE |Found { lv_count } DICOM import jobs.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcefoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

AWS HealthImaging による画像セットへのアクセス

AWS HealthImaging で医療画像データにアクセスするには、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。

[重要]

インポート中、HealthImaging は DICOM インスタンスバイナリ (.dcm ファイル) を処理し、画像セットに変換します。HealthImaging [クラウドネイティブアクション](#) (APIs) を使用して、データストアとイメージセットを管理します。HealthImaging の [DICOMweb サービス表現](#) を使用して DICOMweb レスポンスを返します。

以下のトピックでは AWS マネジメントコンソール、、、 AWS SDKs で HealthImaging クラウドネイティブアクションを使用してイメージセットを検索し AWS CLI、関連するプロパティ、メタデータ、イメージ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](#)

画像セットの理解

画像セットは DICOM シリーズに似ており、AWS HealthImaging の基盤として機能します。画像セットは、DICOM データを HealthImaging にインポートするときに作成されます。サービスは、インポートされた P10 データを、治験、シリーズ、インスタンスの DICOM 階層に従って整理しようとします。

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。

- 重複データや不整合データを永続的に保存して照合するメカニズムを提供します。既にストアにあるプライマリイメージセットと競合するインポートされた P10 データは、非プライマリとして保持されます。メタデータの競合を解決した後、そのデータをプライマリにすることができます。
- 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。

i [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細を説明し、HealthImaging の機能と目的の理解に役立つ例と図を示します。

画像セットとは

画像セットは、DICOM シリーズによく似た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。

i 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。HealthImaging は、受信 DICOM P10 オブジェクトを画像セットにグループ化するときに、次の DICOM 要素を使用します。

画像セットの作成に使用される DICOM 要素

要素名	要素タグ
研究レベルの要素	
Study Date	(0008,0020)
Accession Number	(0008,0050)

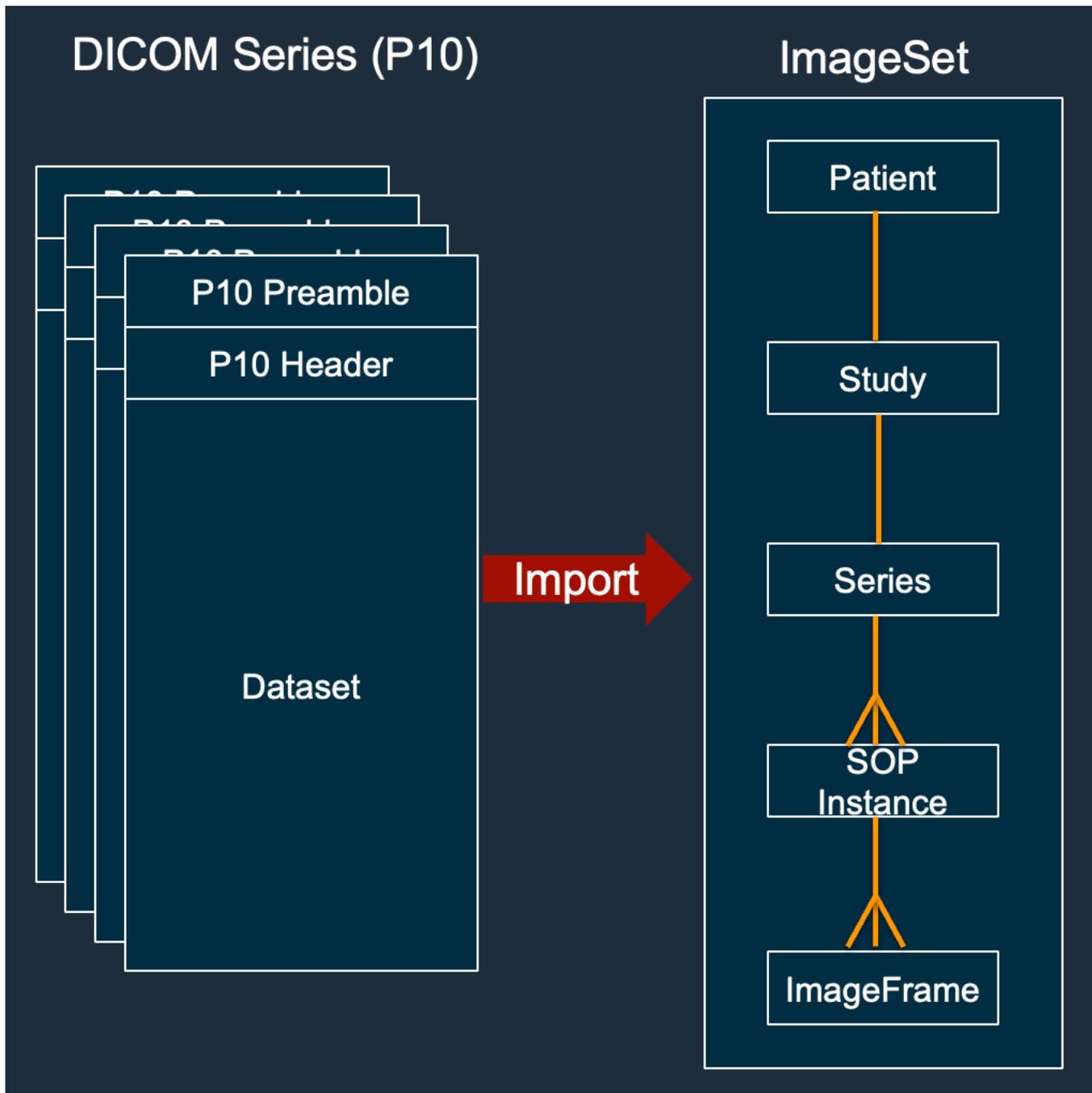
要素名	要素タグ
Patient ID	(0010,0020)
Study Instance UID	(0020,000D)
Study ID	(0020,0010)
シリーズレベルの要素	
Series Instance UID	(0020,000E)
Series Number	(0020,0011)

インポート中、一部のイメージセットは元の転送構文エンコードを保持し、他のイメージセットはデフォルトで高スループット JPEG 2000 (HTJ2K) ロスレスに変換されます。イメージセットが HTJ2K でエンコードされている場合は、表示前にデコードする必要があります。詳細については、「[サポートされる転送構文](#)」および「[イメージフレームデコードライブラリ](#)」を参照してください。

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。

イメージセットは 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:

- 常に新しいイメージセットを作成するか、既存のイメージセットのバージョンを増やしてください。

- SOP インスタンスストレージを重複排除しないでください。同じ SOP インスタンスをインポートするたびに、新しい非プライマリイメージセットまたは既存のプライマリイメージセットの増分バージョンとして追加のストレージが使用されます。
- 整合性のある競合しないメタデータを持つ SOP インスタンスをプライマリイメージセットとして自動的に整理します。これには、整合性のある患者、治験、シリーズメタデータ要素を持つインスタンスが含まれます。
 - DICOM シリーズで構成されるインスタンスが 2 つ以上のインポートジョブにインポートされ、インスタンスがデータストアに既に存在するインスタンスと競合しない場合、すべてのインスタンスは 1 つのプライマリイメージセットに整理されます。
- データストアに既に存在するプライマリイメージセットと競合する DICOM P10 データを含む非プライマリイメージセットを作成します。
- 最後に受信したデータを、プライマリイメージセットの最新バージョンとして保持します。
 - DICOM シリーズで構成されるインスタンスがプライマリイメージセットであり、1 つのインスタンスが再度インポートされると、新しいコピーがプライマリイメージセットに挿入され、バージョンが増分されます。

イメージ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、イメージセットメタデータを取得します。返されるメタデータは、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
              }
            ]
          }
        }
      }
    }
  }
}
```

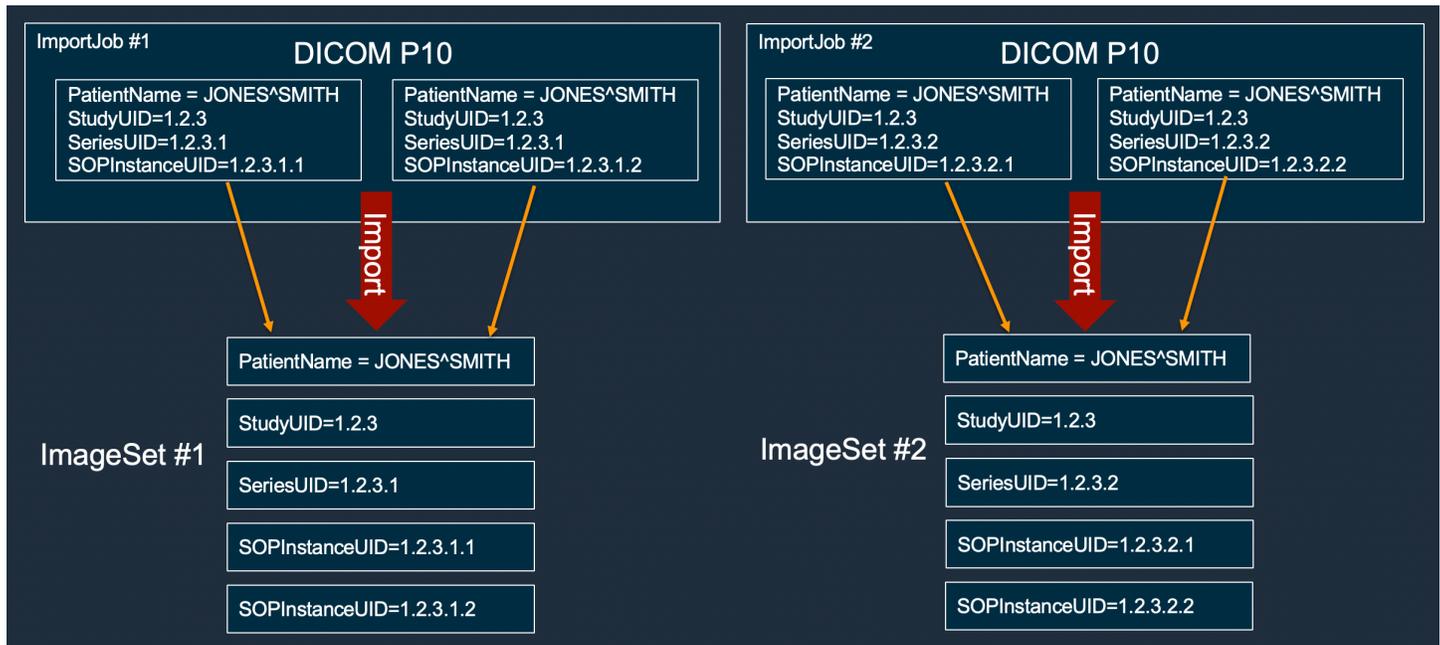
```

}
}

```

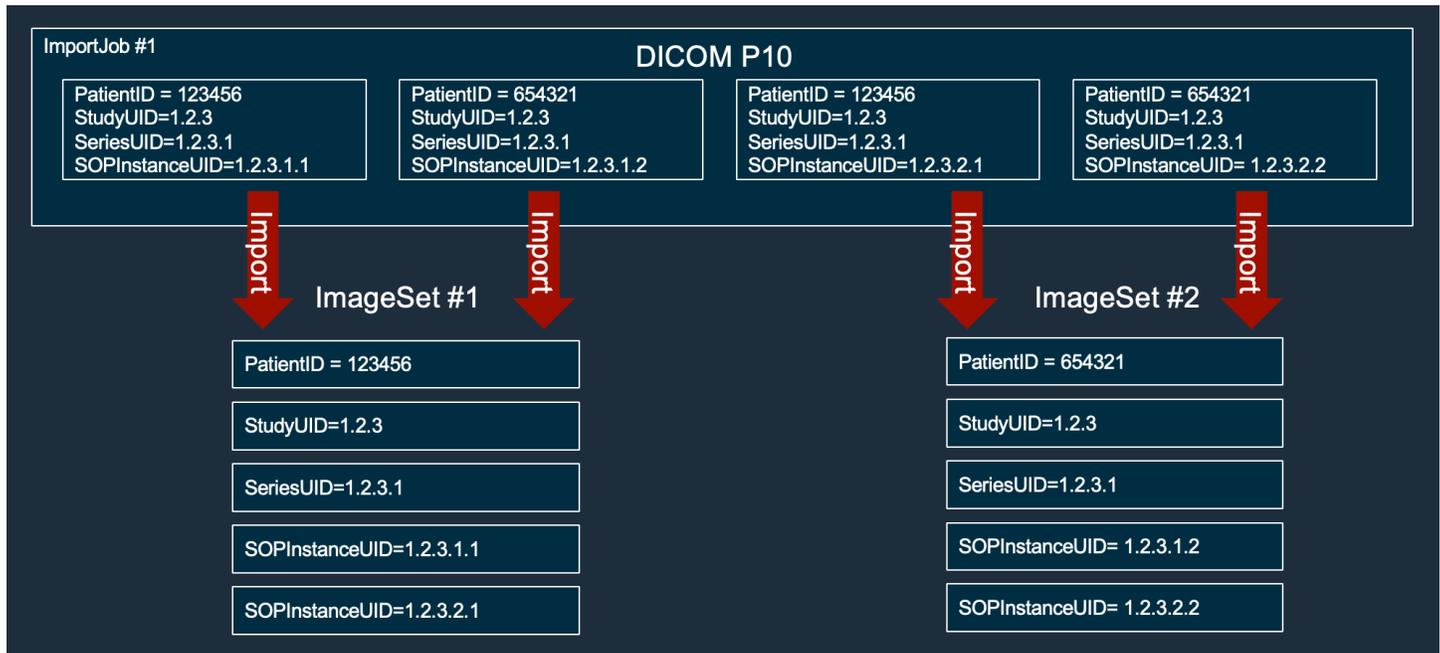
画像セットの作成例: 複数のインポートジョブ

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。



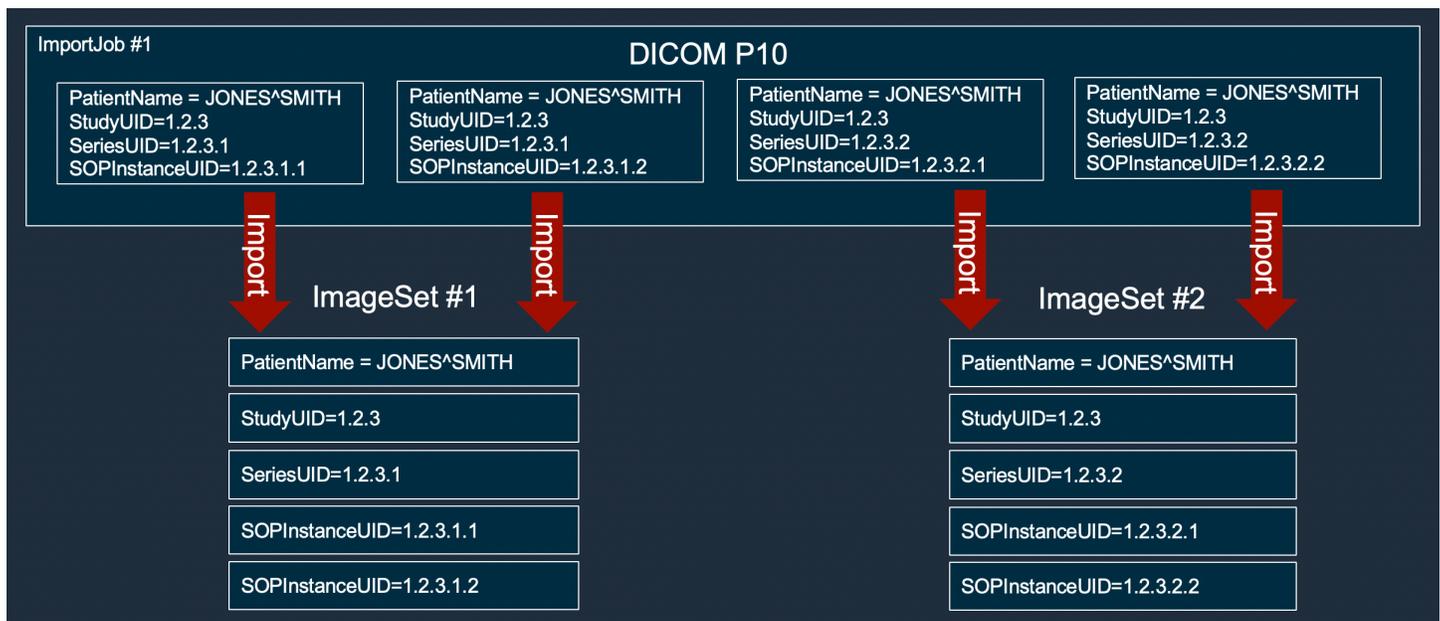
画像セットの作成例: 2つのバリエーションを含む1つのインポートジョブ

次の例は、インスタンス1と3にインスタンス2と4とは異なる患者IDsがあるため、単一のイメージセットへのマージに失敗する単一のインポートジョブを示しています。これを解決するには、UpdateImageSetMetadataアクションを使用して、既存のプライマリイメージセットと患者IDの競合を解決できます。競合が解決されたら、引数でCopyImageSetアクションを使用して--promoteToPrimary、イメージセットをプライマリイメージセットに追加できます。



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、ACTIVEHealthImaging データストア内のすべての [イメージセット](#) に対して検索クエリを実行します。以下のメニューでは、 の手順 AWS マネジメント

コンソール と、 AWS CLI および 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

イメージ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、最新から古い順にソートします。
- 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。

イメージセットを検索するには

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

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イメージ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イメージセットを検索する

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 (貼り付け) します。

 Note

シリーズインスタンス UID 値は、[DICOM 一意識別子 \(UIDs\) のレジストリ](#)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、それらの間に少なくとも1つの期間を含む一連の数字が含まれていることに注意してください。期間をシリーズインスタンスUIDsの最初または最後に使用することはできません。文字と空白は使用できません。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] を選択してください。

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イメージセットを検索する

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] を選択してください。

選択した日付範囲内にある画像セットは、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++

画像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 #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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して、updatedAt フィールドで ASC 順序にレスポンスをソートします。

```
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、AWS SDK for C++ API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 : DICOMStudyDate と DICOMStudyTime を使用することで、BETWEEN 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、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 : DICOMStudyTime は選択可能です。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 (スタデイ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の間で、HealthImaging に保持されている DICOM スタデイを含む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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して画像セットを検索し、updatedAt フィールドのレスポンスを ASC 順序でソートするには

次の `search-image-sets` コード例では、`DICOMSeriesInstanceUID` で `EQUAL` 演算子を使用し、`updatedAt` で `BETWEEN` 演算子を使用して画像セットを検索し、`updatedAt` フィールドのレスポンスを `ASC` 順序でソートします。

注: `updatedAt` 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": [{  
      "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 Developer Guide」の「[Searching image sets](#)」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「[SearchImageSets](#)」を参照してください。

Java

SDK for Java 2.x

画像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: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```
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()

.dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                        .dicomStudyDate((LocalDate.now()
                        .format(formatter)))
                        .dicomStudyTime("000000.000")
                        .build())
                        .build())
                        .build(),
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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して、updatedAt フィールドで ASC 順序にレスポンスをソートします。

```
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、AWS SDK for Java 2.x API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

画像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: [{ DICOPatientId: "1234567" }],
        operator: "EQUAL",
      },
    ],
  };

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

ユースケース #2: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```
const datastoreId = "12345678901234567890123456789012";

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

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

```
    },
  ],
  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: `DICOMSeriesInstanceUID` で EQUAL 演算子を使用し、`updatedAt` で BETWEEN 演算子を使用して、`updatedAt` フィールドで ASC 順序にレスポンスをソートします。

```
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、AWS SDK for JavaScript API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

画像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: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```
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\nDICOMStudyTime\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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して、updatedAt フィールドで ASC 順序にレスポンスをソートします。

```
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、AWS SDK for Python (Boto3) API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->searchimagesets(  
    iv_datastoreid = iv_datastore_id  
    io_searchcriteria = io_search_criteria ).  
  DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).  
  DATA(lv_count) = lines( lt_imagesets ).  
  MESSAGE |Found { lv_count } image sets.| TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcefoundex.  
  MESSAGE 'Resource not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[SearchImageSets](#)を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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、HealthImaging で指定された [イメージセット](#) のプロパティを返します。以下のメニューでは、 の手順 AWS マネジメントコンソール と、 AWS CLI および 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 HealthImaging は画像セットの最新バージョンのプロパティを返します。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。

DICOMwebウェブサービスの GetDICOMInstance HealthImaging 表現である `dicomweb` を使用して、DICOM インスタンスバイナリ (.dcm ファイル) を返します。詳細については、「[HealthImaging から DICOM インスタンスを取得する](#)」を参照してください。

画像セットのプロパティを取得するには

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

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 Developer Guide」の「[Getting image set properties](#)」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「[GetImageSet](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[GetImageSet](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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 = "xxxxxxxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxxxxxx",
    imageSetVersion = "",
) => {
```


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、AWS SDK for Python (Boto3) API リファレンスの「[GetImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_image_set_id = '1234567890123456789012345678901234567890'
  " iv_version_id = '1' (optional)
  IF iv_version_id IS NOT INITIAL.
    oo_result = lo_mig->getimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id
      iv_versionid = iv_version_id ).
  ELSE.
    oo_result = lo_mig->getimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
  ENDIF.
  DATA(lv_state) = oo_result->get_imagesetstate( ).
  MESSAGE |Image set retrieved with state: { lv_state }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
```

```
MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[GetImageSet](#)を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

画像セットメタデータの取得

GetImageSetMetadata アクションを使用して、HealthImaging で指定された[イメージセット](#)の[メタデータ](#)を取得します。以下のメニューでは、手順と、AWS マネジメントコンソール、AWS CLI および 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、HealthImaging は画像セットの最新バージョンのメタデータ属性を返します。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。

インポート後も大きな画像セットメタデータが処理中の場合は、409 が返され `ConflictException` することがあります。処理が完了したら、数秒後にリクエストを再試行します。

`GetDICOMInstanceMetadataHealthImaging` の DICOMweb サービスの表現を使用して、DICOM インスタンスメタデータ (.json ファイル) を返します。詳細については、「[HealthImaging からの DICOM インスタンスメタデータの取得](#)」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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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 outputPath: 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, outputPath, clientConfig))
{
    std::cout << "Successfully retrieved image set metadata." <<
std::endl;
    std::cout << "Metadata stored in: " << outputPath << std::endl;
}
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Getting image set metadata](#)」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「[GetImageSetMetadata](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[GetImageSetMetadata](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 "node: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 = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  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、AWS SDK for JavaScript API リファレンスの「[GetImageSetMetadata](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[GetImageSetMetadata](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_image_set_id = '1234567890123456789012345678901234567890'
  " iv_version_id = '1' (optional)
  IF iv_version_id IS NOT INITIAL.
    oo_result = lo_mig->getimagesetmetadata(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id
      iv_versionid = iv_version_id ).
  ELSE.
    oo_result = lo_mig->getimagesetmetadata(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
  ENDIF.
  DATA(lv_metadata_blob) = oo_result->get_imagesetmetadatablob( ).
  MESSAGE 'Image set metadata retrieved.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[GetImageSetMetadata](#)を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

① 可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

転送構文メタデータ

DICOM データをインポートする場合、HealthImaging は画像セットメタデータの転送構文属性の元の値を保持します。インポートされた元の DICOM データの転送構文は `TransferSyntaxUID` として保存されます。HealthImaging は `StoredTransferSyntaxUID` を使用して、データストア内のイメージフレームデータのエンコードに使用される形式を示します。HTJ2K 1.2.840.10008.1.2.4.202 対応データストア (デフォルト) の場合は、JPEG 2000 Lossless 対応データストア 1.2.840.10008.1.2.4.90 の場合はです。

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、HealthImaging の特定のイメージセットの HTJ2K-encoded またはネイティブ JPEG 2000 ロスレスイメージフレームを取得します。[???](#)次のメニューでは、AWS CLI および AWS SDKs。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。

① Note

GetImageFrame アクションを使用するときは、次の点に注意してください。

- [インポート](#)中、HealthImaging は一部の転送構文のエンコードを保持し、他の転送構文を HTJ2K 可逆 (デフォルト) または JPEG 2000 可逆に変換します。GetImageFrame アクションは、インスタンスのストアド転送構文でイメージフレームを返します。取り出しレイテンシーを最小限に抑えるために、取り出し中にトランスコードは実行されません。転送構文によっては、イメージビューワーで表示する前にイメージフレームをデコードする必要がある場合があります。詳細については、「[サポートされる転送構文](#)」および「[イメージフレームデコードライブラリ](#)」を参照してください。
- HealthImaging に保存されているインスタンスで、転送構文の MPEG ファミリー (MPEG2, MPEG-4 AVC/H.264、HEVC/H.265 を含む) でエンコードされた 1 つ以上のイメージフレームがある場合、GetImageFrame アクションは [保存された転送構文](#) でビデオオブジェクトを返します。

- イメージフレームの転送構文は、Content-Type HTTPヘッダーレスポンス要素で指定されます。たとえば、HTJ2K でエンコードされたイメージフレームには `Content-Type: image/jph` header が含まれます。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。
- `GetDICOMInstanceFramesHealthImaging` が DICOMweb サービスを表す `HealthImaging` を使用して、DICOMweb 互換ビューワーとアプリケーションの DICOMweb インスタンスフレーム (multipart リクエスト) を取得することもできます。詳細については、「[HealthImaging からの DICOM インスタンスフレームの取得](#)」を参照してください。

画像セットのピクセルデータを取得するには

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

AWS コンソール

Note

AWS マネジメントコンソール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、[イメージフレームデコードライブラリ](#)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、AWS SDK for C++ API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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.jpg
```

注：このコード例には出力は含まれていません。GetImageFrame という操作は、ピクセルデータのストリームを `imageframe.jpg` ファイルに返すからです。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 Developer Guide」の「[Getting image set pixel data](#)」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「[GetImageFrame](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for JavaScript API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  " iv_image_set_id = '1234567890123456789012345678901234567890'  
  " iv_image_frame_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->getimageframe(  
    iv_datastoreid = iv_datastore_id  
    iv_imagesetid = iv_image_set_id  
    io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(  
      iv_imageframeid = iv_image_frame_id ) ).  
  DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).  
  MESSAGE 'Image frame retrieved.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.
```

```
MESSAGE 'Image frame not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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

[重要]

インポート中、HealthImaging は DICOM インスタンスバイナリ (.dcm ファイル) を処理し、画像セットに変換します。HealthImaging [クラウドネイティブアクション](#) (APIs) を使用して、データストアとイメージセットを管理します。HealthImaging [の DICOMweb サービス表現](#)を使用して、DICOMweb レスポンスを返します。

HealthImaging には、イメージセットの変更プロセスを簡素化するためのクラウドネイティブ APIs がいくつか用意されています。以下のトピックでは、AWS CLI と AWS SDKs を使用してイメージ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](#)
- [画像セットメタデータの更新](#)
- [画像セットのコピー](#)
- [画像セットの削除](#)

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、HealthImaging で[イメージセット](#)のバージョン履歴を一覧表示します。以下のメニューでは、 の手順 AWS マネジメントコンソール と、AWS CLI および 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 HealthImaging は画像セットに加えられたすべての変更を記録します。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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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 Developer Guide」の「[Listing image set versions](#)」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「[ListImageSetVersions](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[ListImageSetVersions](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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,
  );

  const 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      imageSetState: 'ACTIVE',
//      versionId: '1'
//    }]
// }
return imageSetPropertiesList;
};
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[ListImageSetVersions](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->listimagesetversions(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id ).
```

```
DATA(lt_versions) = oo_result->get_imagesetpropertieslist( ).
DATA(lv_count) = lines( lt_versions ).
MESSAGE |Found { lv_count } image set versions.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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 HealthImaging のイメージセット [メタデータ](#) を更新します。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

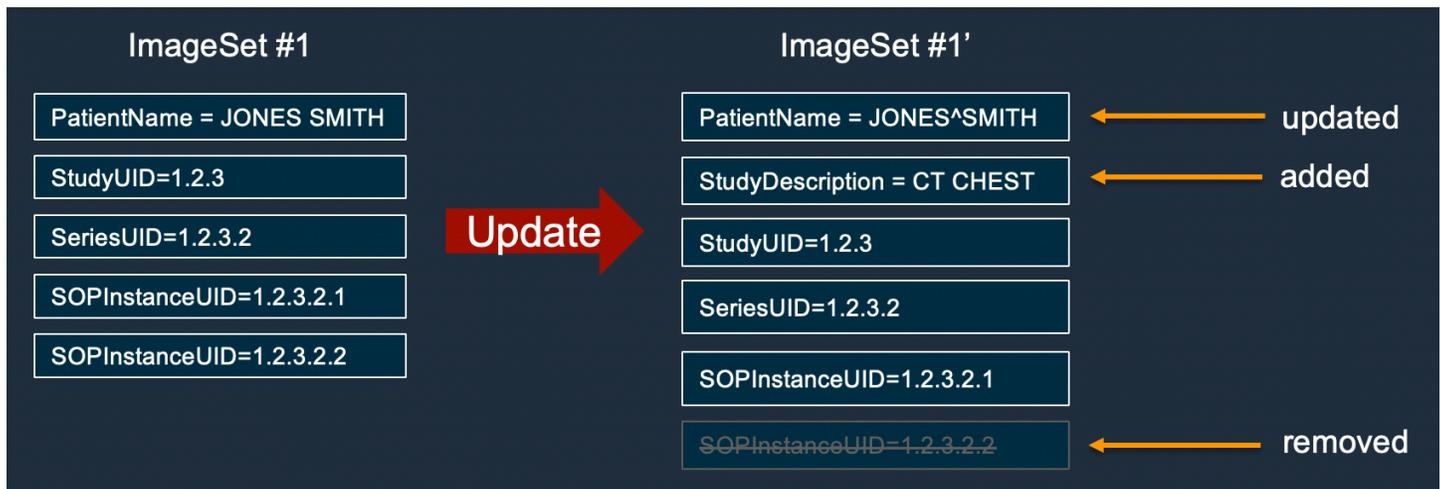
化したりすることもできます。詳細については、「AWS HealthImaging API リファレンス」の「[UpdateImageSetMetadata](#)」を参照してください。

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。イメージセット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。以前のイメージセットバージョン ID に戻すには、オプションの [revertToVersionId](#) パラメータを使用します。
- 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、更新中のイメージセットのそれぞれの状態とステータスを提供するために使用できます。LOCKED イメージセットに対して他の書き込みオペレーションを実行することはできません。
- UpdateImageSetMetadata アクションが成功しない場合は、[message](#) レスポンス要素を確認し、[common errors](#) を確認します。
- DICOM 要素の制約はメタデータの更新に適用されます。[force](#) リクエストパラメータを使用すると、[上書き](#)する場合に、プライマリ以外の [イメージセット](#) の要素を更新できません [DICOM メタデータの制約](#)。
- 患者レベルとシリーズレベルのメタデータ要素は、プライマリ [イメージセット](#) では更新できません。UpdateImageSet は、プライマリ [イメージセット](#) の StudyInstanceUID、SeriesInstanceUID、SOPInstanceUID を更新する --force をサポートしていません。
- [force](#) リクエストパラメータを設定して、プライマリ以外の [イメージセット](#) で UpdateImageSetMetadata アクションを強制的に完了します。このパラメータを設定すると、イメージセットに次の更新が許可されます。
 - Tag.StudyInstanceUID、Tag.SeriesInstanceUID、Tag.SOPInstanceUID および Tag.StudyID 属性の更新
 - インスタンスレベルのプライベート DICOM データ要素の追加、削除、または更新
- イメージセットをプライマリに昇格させるアクションにより、イメージセット ID が変更されます。

以下の図は、HealthImaging で更新される画像セットメタデータを表しています。



画像セットのメタデータを更新するには

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

AWS CLI および SDKs

CLI

AWS CLI

例 1: 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` の例では、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 \
  --cli-binary-format raw-in-base64-out \
  --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json` の内容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{\"PatientName\":\"MX^MX\"}}}"
  }
}
```

```
}
```

出力:

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

例 2: 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` の例では、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 \
  --cli-binary-format raw-in-base64-out \
  --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json` の内容

```
{
  "DICOMUpdates": {
    "removableAttributes": "{\"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"
}
```

例 3: 画像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` の例では、画像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 \
  --cli-binary-format raw-in-base64-out \
  --update-image-set-metadata-updates file://metadata-updates.json \
  --force
```

`metadata-updates.json` の内容

```
{
  "DICOMUpdates": {
    "removableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}}"
  }
}
```

出力:

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

例 4: 画像セットを以前のバージョンに戻すには

次の `update-image-set-metadata` の例は、画像セットを以前のバージョンに戻す方法を示しています。CopyImageSet および UpdateImageSetMetadata アクションは、新しいバージョンの画像セットを作成します。

```
aws medical-imaging update-image-set-metadata \  
  --datastore-id 12345678901234567890123456789012 \  
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \  
  --latest-version-id 3 \  
  --cli-binary-format raw-in-base64-out \  
  --update-image-set-metadata-updates '{"revertToVersionId": "1"}'
```

出力:

```
{  
  "datastoreId": "12345678901234567890123456789012",  
  "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",  
  "latestVersionId": "4",  
  "imageSetState": "LOCKED",  
  "imageSetWorkflowStatus": "UPDATING",  
  "createdAt": 1680027126.436,  
  "updatedAt": 1680042257.908  
}
```

例 5: インスタンスにプライベート DICOM データ要素を追加するには

次の `update-image-set-metadata` の例は、画像セット内で指定されたインスタンスにプライベート要素を追加する方法を示しています。DICOM 標準では、標準データ要素に含めることができない情報の通信に、プライベートデータ要素が許可されます。UpdateImageSetMetadata アクションを使用して、プライベートデータ要素を作成、更新、削除できます。

```
aws medical-imaging update-image-set-metadata \  
  --datastore-id 12345678901234567890123456789012 \  
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \  
  --latest-version-id 1 \  
  --cli-binary-format raw-in-base64-out \  
  --force \  
  --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\": {\"001910F9\": \"97\"}, \"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}}"
  }
}
```

出力:

```
{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
  "updatedAt": 1680042257.908,
  "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
  "imageSetState": "LOCKED",
  "createdAt": 1680027126.436,
  "datastoreId": "12345678901234567890123456789012"
}
```

例 6: プライベート DICOM データ要素をインスタンスに対して更新するには

次の `update-image-set-metadata` の例は、画像セット内のインスタンスに属するプライベートデータ要素の値を更新する方法を示しています。

```
aws medical-imaging update-image-set-metadata \
  --datastore-id 12345678901234567890123456789012 \
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
  --latest-version-id 1 \
  --cli-binary-format raw-in-base64-out \
  --force \
  --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json` の内容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\": {\"00091001\": \"GE_GENESIS_DD\"}}}}}}}"
  }
}
```

```
}
}
```

出力:

```
{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
  "updatedAt": 1680042257.908,
  "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
  "imageSetState": "LOCKED",
  "createdAt": 1680027126.436,
  "datastoreId": "12345678901234567890123456789012"
}
```

例 7: force パラメータを使用して SOPInstanceUID を更新するには

次の update-image-set-metadata の例は、force パラメータを使用して SOPInstanceUID を更新し、DICOM メタデータの制約をオーバーライドする方法を示しています。

```
aws medical-imaging update-image-set-metadata \
  --datastore-id 12345678901234567890123456789012 \
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
  --latest-version-id 1 \
  --cli-binary-format raw-in-base64-out \
  --force \
  --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\":1.1,\"Study\":{\"Series\":{\"1.3.6.1.4.1.5962.99.1.3633258862.2104868982.1369432891697.3656.0\":{\"Instances\":{\"1.3.6.1.4.1.5962.99.1.3633258862.2104868982.1369432891697.3659.0\":{\"DICOM\":{\"SOPInstanceUID\":{\"1.3.6.1.4.1.5962.99.1.3633258862.2104868982.1369432891697.3659.9\"}}}}}}}"
  }
}
```

出力:

```
{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
  "updatedAt": 1680042257.908,
  "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
  "imageSetState": "LOCKED",
  "createdAt": 1680027126.436,
  "datastoreId": "12345678901234567890123456789012"
}
```

詳細については、「AWS HealthImaging Developer Guide」の「[Updating image set metadata](#)」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「[UpdateImageSetMetadata](#)」を参照してください。

Java

SDK for Java 2.x

```
/**
 * Update the metadata of an AWS HealthImaging image set.
 *
 * @param medicalImagingClient - The AWS HealthImaging client object.
 * @param datastoreId          - The datastore ID.
 * @param imageSetId          - The image set ID.
 * @param versionId           - The version ID.
 * @param metadataUpdates     - A MetadataUpdates object containing the
updates.
 * @param force                - The force flag.
 * @throws MedicalImagingException - Base exception for all service
exceptions thrown by AWS HealthImaging.
 */
public static void updateMedicalImageSetMetadata(MedicalImagingClient
medicalImagingClient,
                                                String datastoreId,
                                                String imageSetId,
                                                String versionId,
                                                MetadataUpdates
metadataUpdates,
                                                boolean force) {
```

```
try {
    UpdateImageSetMetadataRequest updateImageSetMetadataRequest =
UpdateImageSetMetadataRequest
        .builder()
        .datastoreId(datastoreId)
        .imageSetId(imageSetId)
        .latestVersionId(versionId)
        .updateImageSetMetadataUpdates(metadataUpdates)
        .force(force)
        .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());
    throw e;
}
}
```

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()
        .updateableAttributes(SdkBytes.fromByteBuffer(
            ByteBuffer.wrap(insertAttributes
                .getBytes(StandardCharsets.UTF_8))))
        .build())
    .build();
```


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.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  
```

```
//   imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   imageSetState: 'LOCKED',
//   imageSetWorkflowStatus: 'UPDATING',
//   latestVersionId: '4',
//   updatedAt: 2023-09-27T19:41:43.494Z
// }
return response;
} catch (err) {
  console.error(err);
}
};
```

ユースケース 1: 属性を挿入または更新し、強制的に更新します。

```
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,
  true,
);
```

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,  
);
```

ユースケース 4: 以前のバージョンに戻します。

```
const updateMetadata = {  
    revertToVersionId: "1",  
};  
  
await updateImageSetMetadata(  
    datastoreID,  
    imageSetID,  
    versionID,  
    updateMetadata,  
);
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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, force=False
):
    """
    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": {"updateableAttributes":
            {"\SchemaVersion\":1.1,\Patient\":{"DICOM\":{"PatientName\":
"\Garcia^Gloria\}}}}}"}
    :param force: Force the update.
    :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,
            force=force,
        )
    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, force
)
```

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, force
)
```

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": {
```

```
"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}  
    }  
    }  
    }  
}""  
metadata = {"DICOMUpdates": {"removableAttributes": attributes}}  
  
self.update_image_set_metadata(  
    data_store_id, image_set_id, version_id, metadata, force  
)
```

ユースケース 4: 以前のバージョンに戻します。

```
metadata = {"revertToVersionId": "1"}  
  
self.update_image_set_metadata(  
    data_store_id, image_set_id, version_id, metadata, force  
)
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_datastore_id = '1234567890123456789012345678901234567890'  
    " iv_image_set_id = '1234567890123456789012345678901234567890'
```

```
" iv_latest_version_id = '1'
" iv_force = abap_false
oo_result = lo_mig->updateimagesetmetadata(
  iv_datastoreid = iv_datastore_id
  iv_imagesetid = iv_image_set_id
  iv_latestversionid = iv_latest_version_id
  io_updateimagesetmetupdates = io_metadata_updates
  iv_force = iv_force ).
DATA(lv_new_version) = oo_result->get_latestversionid( ).
MESSAGE |Image set metadata updated to version: { lv_new_version }.| TYPE
'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
  MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[UpdateImageSetMetadata](#)を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

4. 引数で CopyImageSet アクション `--promoteToPrimary` を使用して、更新されたイメージセットをプライマリコレクションに追加します。

```
aws medical-imaging copy-image-set --datastore-id a8d19e7875e1532d9b5652f6b25e12c9 --source-image-set-id 103785414bc2c89330f7ce51bbd13f7a --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "2" }}' --region us-west-2 --promote-to-primary
```

5. プライマリ以外のイメージセットを削除します。

```
aws medical-imaging delete-image-set --datastore-id a8d19e7875e1532d9b5652f6b25e12c9 --image-set-id 103785414bc2c89330f7ce51bbd13f7a
```

プライマリ以外のイメージセットをプライマリにするには

1. UpdateImageSetMetadata アクションを使用して、既存のプライマリイメージセットとの競合を解決します。

```
aws medical-imaging update-image-set-metadata \
  --region us-west-2 \
  --datastore-id a8d19e7875e1532d9b5652f6b25e12c9 \
  --image-set-id 103785414bc2c89330f7ce51bbd13f7a \
  --latest-version-id 1 \
  --cli-binary-format raw-in-base64-out \
  --update-image-set-metadata-updates '{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{
      \"PatientID\":\"1234\"}}}"
  }
}'
```

2. 競合が解決したら、CopyImageSet アクションを引数とともに使用 `--promoteToPrimary` して、イメージセットをプライマリイメージセットコレクションに追加します。

```
aws medical-imaging copy-image-set --datastore-id a8d19e7875e1532d9b5652f6b25e12c9 --source-image-set-id 103785414bc2c89330f7ce51bbd13f7a --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "2" }}' --region us-west-2 --
```

```
promote-to-primary
```

- CopyImageSet アクションが成功したことを確認したら、ソースの非プライマリイメージセットを削除します。

```
aws medical-imaging delete-image-set --datastore-  
id a8d19e7875e1532d9b5652f6b25e12c9 --image-set-  
id 103785414bc2c89330f7ce51bbd13f7a
```

画像セットのコピー

CopyImageSet アクションを使用して、HealthImaging で [イメージセット](#) をコピーします。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。新しいイメージセットにコピーしてイメージ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。

Note

CopyImageSet アクションを使用するときは、次の点に注意してください。

- CopyImageSet アクションは、新しいイメージセット、またはの新しいバージョンを作成します `destinationImageSet`。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。
- 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、ロックされたイメージ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。ロックされたイメージセットでは、他の書き込みオペレーションを実行できません。
- CopyImageSet では、SOP インスタンスUIDs はイメージセット内で一意である必要があります。
- を使用して SOP インスタンスのサブセットをコピーできます `copiableAttributes`。これにより、から1つ以上のSOP インスタンスを選択して `sourceImageSet`、にコピーできます `destinationImageSet`。
- CopyImageSet アクションが成功しない場合は、を呼び出し `getImageSet` で `message` プロパティを確認します。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 アクションではdestinationImageSet、オプションの [force](#) パラメータが指定されていない限り、sourceImageSetとに一貫したメタデータが必要です。
- [force](#) パラメータを設定して、sourceImageSetと の間に一貫性のないメタデータ要素がある場合でも、オペレーションを強制しますdestinationImageSet。このような場合、患者、治験、シリーズメタデータは で変更されませんdestinationImageSet。

画像セットをコピーするには

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

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 の例では、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` の例では、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"
}

```

```
}

```

例 3: ソース画像セットからコピー先画像セットにインスタンスのサブセットをコピーするには。

次の `copy-image-set` の例では、ソース画像セットからコピー先画像セットに 1 つの DICOM インスタンスをコピーします。force パラメータは、患者、検査、シリーズレベルの属性の不整合を上書きするために提供されます。

```
aws medical-imaging copy-image-set \
  --datastore-id 12345678901234567890123456789012 \
  --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
  --copy-image-set-information '{"sourceImageSet":
{"latestVersionId": "1", "DICOMCopies": {"copiableAttributes":
{"SchemaVersion": "1.1", "Study": {"Series":
{"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3666.0":
{"Instances":
{"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3669.0":
}}}}}}}', "destinationImageSet": {"imageSetId":
"b9eb50d8ee682eb9fcf4acbf92f62bb7", "latestVersionId": "1"}}' \
  --force
```

出力:

```
{
  "destinationImageSetProperties": {
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "COPYING",
    "updatedAt": 1680042505.135,
    "imageSetId": "b9eb50d8ee682eb9fcf4acbf92f62bb7",
    "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 Developer Guide」の「[Copying an image set](#)」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「[CopyImageSet](#)」を参照してください。

Java

SDK for Java 2.x

```
/**
 * Copy an AWS HealthImaging image set.
 *
 * @param medicalImagingClient - The AWS HealthImaging client object.
 * @param datastoreId          - The datastore ID.
 * @param imageSetId          - The image set ID.
 * @param latestVersionId     - The version ID.
 * @param destinationImageSetId - The optional destination image set ID,
 ignored if null.
 * @param destinationVersionId - The optional destination version ID,
 ignored if null.
 * @param force                - The force flag.
 * @param subsets              - The optional subsets to copy, ignored if
 null.
 * @return                     - The image set ID of the copy.
 * @throws MedicalImagingException - Base exception for all service
 exceptions thrown by AWS HealthImaging.
 */
public static String copyMedicalImageSet(MedicalImagingClient
medicalImagingClient,
                                         String datastoreId,
                                         String imageSetId,
                                         String latestVersionId,
                                         String destinationImageSetId,
                                         String destinationVersionId,
                                         boolean force,
                                         Vector<String> subsets) {

    try {
```

```
CopySourceImageSetInformation.Builder copySourceImageSetInformation =
CopySourceImageSetInformation.builder()
    .latestVersionId(latestVersionId);

// Optionally copy a subset of image instances.
if (subsets != null) {
    String subsetInstanceToCopy =
getCopiableAttributesJSON(imageSetId, subsets);

copySourceImageSetInformation.dicomCopies(MetadataCopies.builder()
    .copiableAttributes(subsetInstanceToCopy)
    .build());
}

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

// Optionally designate a destination image set.
if (destinationImageSetId != null) {
    copyImageSetBuilder =
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()
    .imageSetId(destinationImageSetId)
    .latestVersionId(destinationVersionId)
    .build());
}

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

CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

コピー可能な属性を作成するユーティリティ関数。

```
/**
 * Create a JSON string of copiable image instances.
 *
 * @param imageSetId - The image set ID.
 * @param subsets    - The subsets to copy.
 * @return A JSON string of copiable image instances.
 */
private static String getCopiableAttributesJSON(String imageSetId,
Vector<String> subsets) {
    StringBuilder subsetInstanceToCopy = new StringBuilder(
        ""
        {
            "SchemaVersion": 1.1,
            "Study": {
                "Series": {
                    ""
                }
            }
        }
    );

    subsetInstanceToCopy.append(imageSetId);

    subsetInstanceToCopy.append(
        ""
        {
            "Instances": {
                ""
            }
        }
    );

    for (String subset : subsets) {
        subsetInstanceToCopy.append("'" + subset + "\" : {},");
    }
    subsetInstanceToCopy.deleteCharAt(subsetInstanceToCopy.length() - 1);
    subsetInstanceToCopy.append("""
        }
    }
}
}
```

```
        """);
    return subsetInstanceToCopy.toString();
}
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
    datastoreId = "xxxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxx",
    sourceVersionId = "1",
    destinationImageSetId = "",
    destinationVersionId = "",
    force = false,
    copySubsets = [],
```

```
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
      copyImageSetInformation: {
        sourceImageSet: { latestVersionId: sourceVersionId },
      },
      force: force,
    };
    if (destinationImageSetId !== "" && destinationVersionId !== "") {
      params.copyImageSetInformation.destinationImageSet = {
        imageSetId: destinationImageSetId,
        latestVersionId: destinationVersionId,
      };
    }

    if (copySubsets.length > 0) {
      let copySubsetsJson;
      copySubsetsJson = {
        SchemaVersion: 1.1,
        Study: {
          Series: {
            imageSetId: {
              Instances: {},
            },
          },
        },
      };
    }

    for (let i = 0; i < copySubsets.length; i++) {
      copySubsetsJson.Study.Series.imageSetId.Instances[copySubsets[i]] = {};
    }

    params.copyImageSetInformation.dicomCopies = copySubsetsJson;
  }

  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: 'xxxxxxxxxxxxxxxx',
//    destinationImageSetProperties: {
//      createdAt: 2023-09-27T19:46:21.824Z,
//      imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxxxx',
//      imageSetId: 'xxxxxxxxxxxxxxxx',
//      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:xxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxxxx',
//      imageSetId: 'xxxxxxxxxxxxxxxx',
//      imageSetState: 'LOCKED',
//      imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//      latestVersionId: '4',
//      updatedAt: 2023-09-27T19:46:21.824Z
//    }
//  }
return response;
} catch (err) {
  console.error(err);
}
};
```

コピー先を指定せずにイメージセットをコピーします。

```
await copyImageSet(
  "12345678901234567890123456789012",
  "12345678901234567890123456789012",
  "1",
);
```

コピー先を指定してイメージセットをコピーします。

```
await copyImageSet(  
  "12345678901234567890123456789012",  
  "12345678901234567890123456789012",  
  "1",  
  "12345678901234567890123456789012",  
  "1",  
  false,  
);
```

イメージセットのサブセットを送信先にコピーし、コピーを強制します。

```
await copyImageSet(  
  "12345678901234567890123456789012",  
  "12345678901234567890123456789012",  
  "1",  
  "12345678901234567890123456789012",  
  "1",  
  true,  
  ["12345678901234567890123456789012", "11223344556677889900112233445566"],  
);
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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,
        force=False,
        subsets=[],
    ):
        """
        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.
        :param force: Force the copy.
        :param subsets: The optional subsets to copy. For example:
        ["12345678901234567890123456789012"].
        :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,
```

```

    }
    if len(subsets) > 0:
        copySubsetsJson = {
            "SchemaVersion": "1.1",
            "Study": {"Series": {"imageSetId": {"Instances": {}}}},
        }

        for subset in subsets:
            copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[
                subset
            ] = {}

        copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
            "copiableAttributes": json.dumps(copySubsetsJson)
        }
    copy_results = self.health_imaging_client.copy_image_set(
        datastoreId=datastore_id,
        sourceImageSetId=image_set_id,
        copyImageSetInformation=copy_image_set_information,
        force=force,
    )
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,  
        force=force,  
    )
```

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,  
    force=force,  
)
```

イメージセットのサブセットをコピーします。

```
copy_image_set_information = {  
    "sourceImageSet": {"latestVersionId": version_id}  
}  
  
if len(subsets) > 0:  
    copySubsetsJson = {  
        "SchemaVersion": "1.1",  
        "Study": {"Series": {"imageSetId": {"Instances": {}}}},  
    }  
  
    for subset in subsets:  
        copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]  
[  
            subset  
        ] = {}
```

```

        copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
            "copiableAttributes": json.dumps(copySubsetsJson)
        }

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

```

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「[CopyImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_source_image_set_id = '1234567890123456789012345678901234567890'
    " iv_source_version_id = '1'
    " iv_destination_image_set_id =
'12345678901234567890123456789012345678901234567890' (optional)
    " iv_destination_version_id = '1' (optional)
    " iv_force = abap_false
    DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
        iv_latestversionid = iv_source_version_id ).
    DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(

```

```
io_sourceimageset = lo_source_info ).
IF iv_destination_image_set_id IS NOT INITIAL AND
   iv_destination_version_id IS NOT INITIAL.
   DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
     iv_imagesetid = iv_destination_image_set_id
     iv_latestversionid = iv_destination_version_id ).
   lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
     io_sourceimageset = lo_source_info
     io_destinationimageset = lo_dest_info ).
ENDIF.
oo_result = lo_mig->copyimageset(
  iv_datastoreid = iv_datastore_id
  iv_sourceimagesetid = iv_source_image_set_id
  io_copyimagesetinformation = lo_copy_info
  iv_force = iv_force ).
DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
  MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

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

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

📘 可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

画像セットの削除

HealthImaging で [イメージセット](#) を削除するには、DeleteImageSet アクションを使用します。以下のメニューでは、 の手順 AWS マネジメントコンソール と、AWS CLI および 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 HealthImaging のアクセス設定に基づいてメニューを選択します。

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、AWS SDK for C++ API リファレンスの「[DeleteImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Deleting an image set](#)」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「[DeleteImageSet](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[DeleteImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 = "xxxxxxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  // }
```

```
//     datastoreId: 'xxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxx',
//     imageSetState: 'LOCKED',
//     imageSetWorkflowStatus: 'DELETING'
// }
return response;
};
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[DeleteImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_image_set_id = '1234567890123456789012345678901234567890'
  oo_result = lo_mig->deleteimageset(
    iv_datastoreid = iv_datastore_id
    iv_imagesetid = iv_image_set_id ).
  MESSAGE 'Image set deleted.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
```

```
MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[DeleteImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

AWS HealthImagingによるリソースのタグ付け

メタデータは、タグの形式で HealthImaging リソース ([データストア](#)と[イメージセット](#)) に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または機密データに使用することを目的としたものではありません。

以下のトピックでは、AWS マネジメントコンソール、AWS CLIおよび 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、「[ガイド](#)」の [AWS 「リソースのタグ付け」](#) を参照してください。AWS 全般のリファレンス

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 HealthImaging の[データストア](#)と[イメージセット](#)にタグを付けます。次のコード例は、AWS マネジメントコンソール、AWS CLI、および 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、「[ガイド](#)」の「[AWS リソースのタグ付け](#)」を参照してください。AWS 全般のリファレンス

リソースにタグを付けるには

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

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. [Save changes] (変更の保存) をクリックします。

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「HealthImaging を使用した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、AWS CLI コマンドリファレンスの「[TagResource](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[TagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for JavaScript API リファレンスの「[TagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[TagResource](#)」を参照してください。

 Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_resource_arn = 'arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012'  
  lo_mig->tagresource(  
    iv_resourcearn = iv_resource_arn  
    it_tags = it_tags ).  
  MESSAGE 'Resource tagged successfully.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Resource not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[TagResource](#)」を参照してください。

 Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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 HealthImaging の [データストア](#) と [イメージセット](#) のタグを一覧表示します。次のコード例は、AWS マネジメントコンソール、AWS CLI、および 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、「[ガイド](#)」の [AWS 「リソースのタグ付け](#)」を参照してください。AWS 全般のリファレンス

リソースのタグを一覧表示するには

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

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「HealthImaging を使用した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、「AWS CLI コマンドリファレンス」の「[ListTagsForResource](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[ListTagsForResource](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、「AWS SDK for JavaScript API リファレンス」の「[ListTagsForResource](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[ListTagsForResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
    " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
    oo_result = lo_mig->listtagsforresource( iv_resourcearn =
iv_resource_arn ).
```

```
DATA(lt_tags) = oo_result->get_tags( ).
DATA(lv_count) = lines( lt_tags ).
MESSAGE |Found { lv_count } tags for resource.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[ListTagsForResource](#)を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

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 HealthImaging の[データストア](#)と[イメージセット](#)のタグを解除します。次のコード例は、AWS マネジメントコンソール、AWS CLI、および 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、「[ガイド](#)」の[AWS 「リソースのタグ付け](#)」を参照してください。AWS 全般のリファレンス

リソースのタグを削除するには

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

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. [Save changes] (変更の保存) をクリックします。

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「HealthImaging を使用した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、「AWS CLI Command Reference」の「[UntagResource](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[UntagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、「AWS SDK for JavaScript API リファレンス」の「[UntagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[UntagResource](#)」を参照してください。

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_resource_arn = 'arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012'  
    lo_mig->untagresource(  
        iv_resourcearn = iv_resource_arn  
        it_tagkeys = it_tag_keys ).  
    MESSAGE 'Resource untagged successfully.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Resource not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP [UntagResource](#)」を参照してください。

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

i 可用性の例

必要なものが見つからなかった場合。このページの右側サイドバーにあるフィードバックを提供するリンクを使用して、コード例をリクエストします。

SDK を使用した HealthImaging のコード例 AWS SDKs

次のコード例は、AWS Software Development Kit (SDK) で HealthImaging を使用方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

シナリオは、1つのサービス内から、または他のAWSのサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードの例

- [SDK を使用した HealthImaging の基本的な例 AWS SDKs](#)
 - [Hello HealthImaging](#)
 - [SDK を使用した HealthImaging のアクション AWS SDKs](#)
 - [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](#)
- [SDK を使用した HealthImaging のシナリオ AWS SDKs](#)
 - [AWS SDK を使用して HealthImaging イメージセットとイメージ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 イメージセットのタグ付け](#)

SDK を使用した HealthImaging の基本的な例 AWS SDKs

次のコード例は、 SDKs AWS HealthImaging で AWS の基本を使用する方法を示しています。

例

- [Hello HealthImaging](#)
- [SDK を使用した HealthImaging のアクション AWS SDKs](#)
 - [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 で使用する](#)
- [AWS SDK または CLI UntagResource で使用する](#)
- [AWS SDK または CLI UpdateImageSetMetadata で使用する](#)

Hello HealthImaging

次のコード例では、HealthImaging の使用を開始する方法を示しています。

C++

SDK for C++

CMakeLists.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、AWS SDK for C++ API リファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for JavaScript API リファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 AWS 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 AWS 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、AWS SDK for Python (Boto3) API リファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した HealthImaging のアクション AWS SDKs

次のコード例は、AWS SDKs を使用して個々の HealthImaging アクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

これらは HealthImaging API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。アクションは [SDK を使用した HealthImaging のシナリオ AWS SDKs](#) のコンテキスト内で確認できます。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、「[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 DeletImageSet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

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

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 の例では、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 \  
  --image-set-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 の例では、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"
}

```

例 3: ソース画像セットからコピー先画像セットにインスタンスのサブセットをコピーするには。

次の `copy-image-set` の例では、ソース画像セットからコピー先画像セットに 1 つの DICOM インスタンスをコピーします。force パラメータは、患者、検査、シリーズレベルの属性の不整合を上書きするために提供されます。

```

aws medical-imaging copy-image-set \
  --datastore-id 12345678901234567890123456789012 \
  --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
  --copy-image-set-information '{"sourceImageSet":
{"latestVersionId": "1", "DICOMCopies": {"copiableAttributes":
{"SchemaVersion": "1.1", "Study": {"Series":
{"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3666.0":
{"Instances":
{"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3669.0":
}}}}}}}', "destinationImageSet": {"imageSetId":
"b9eb50d8ee682eb9fcf4acbf92f62bb7", "latestVersionId": "1"}}' \
  --force

```

出力:

```

{
  "destinationImageSetProperties": {
    "latestVersionId": "2",

```

```

        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042505.135,
        "imageSetId": "b9eb50d8ee682eb9fcf4acbf92f62bb7",
        "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 Developer Guide」の「[Copying an image set](#)」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「[CopyImageSet](#)」を参照してください。

Java

SDK for Java 2.x

```

/**
 * Copy an AWS HealthImaging image set.
 *
 * @param medicalImagingClient - The AWS HealthImaging client object.
 * @param datastoreId          - The datastore ID.
 * @param imageSetId          - The image set ID.
 * @param latestVersionId     - The version ID.
 * @param destinationImageSetId - The optional destination image set ID,
 ignored if null.
 * @param destinationVersionId - The optional destination version ID,
 ignored if null.
 * @param force                - The force flag.
 * @param subsets              - The optional subsets to copy, ignored if
 null.
 * @return                     - The image set ID of the copy.

```

```
* @throws MedicalImagingException - Base exception for all service
exceptions thrown by AWS HealthImaging.
*/
public static String copyMedicalImageSet(MedicalImagingClient
medicalImagingClient,
                                         String datastoreId,
                                         String imageSetId,
                                         String latestVersionId,
                                         String destinationImageSetId,
                                         String destinationVersionId,
                                         boolean force,
                                         Vector<String> subsets) {

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

        // Optionally copy a subset of image instances.
        if (subsets != null) {
            String subsetInstanceToCopy =
getCopiableAttributesJSON(imageSetId, subsets);

copySourceImageSetInformation.dicomCopies(MetadataCopies.builder()
            .copiableAttributes(subsetInstanceToCopy)
            .build());
        }

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

        // Optionally designate a destination image set.
        if (destinationImageSetId != null) {
            copyImageSetBuilder =
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()
            .imageSetId(destinationImageSetId)
            .latestVersionId(destinationVersionId)
            .build());
        }

        CopyImageSetRequest copyImageSetRequest =
CopyImageSetRequest.builder()
            .datastoreId(datastoreId)
```

```

        .sourceImageSetId(imageSetId)
        .copyImageSetInformation(copyImageSetBuilder.build())
        .force(force)
        .build();

        CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

```

コピー可能な属性を作成するユーティリティ関数。

```

/**
 * Create a JSON string of copiable image instances.
 *
 * @param imageSetId - The image set ID.
 * @param subsets    - The subsets to copy.
 * @return A JSON string of copiable image instances.
 */
private static String getCopiableAttributesJSON(String imageSetId,
Vector<String> subsets) {
    StringBuilder subsetInstanceToCopy = new StringBuilder(
        """"
        {
            "SchemaVersion": 1.1,
            "Study": {
                "Series": {
                    """"
                }
            }
        }
        """"
    );

    subsetInstanceToCopy.append(imageSetId);

    subsetInstanceToCopy.append(
        """"
        ": {

```

```
        "Instances": {
            ""
        };

        for (String subset : subsets) {
            subsetInstanceToCopy.append("'" + subset + "\": {},");
        }
        subsetInstanceToCopy.deleteCharAt(subsetInstanceToCopy.length() - 1);
        subsetInstanceToCopy.append("''"
            }
        }
    }
}
return subsetInstanceToCopy.toString();
}
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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.
* @param {boolean} force - Force the copy action.
* @param {[string]} copySubsets - A subset of instance IDs to copy.
*/
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = [],
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
      copyImageSetInformation: {
        sourceImageSet: { latestVersionId: sourceVersionId },
      },
      force: force,
    };
    if (destinationImageSetId !== "" && destinationVersionId !== "") {
      params.copyImageSetInformation.destinationImageSet = {
        imageSetId: destinationImageSetId,
        latestVersionId: destinationVersionId,
      };
    }

    if (copySubsets.length > 0) {
      let copySubsetsJson;
      copySubsetsJson = {
        SchemaVersion: 1.1,
        Study: {
          Series: {
            imageSetId: {
              Instances: {},
            },
          },
        },
      };
    }
  };
}
```

```
for (let i = 0; i < copySubsets.length; i++) {
  copySubsetsJson.Study.Series.imageSetId.Instances[copySubsets[i]] = {};
}

params.copyImageSetInformation.dicomCopies = copySubsetsJson;
}

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: 'xxxxxxxxxxxxxxxx',
//   destinationImageSetProperties: {
//     createdAt: 2023-09-27T19:46:21.824Z,
//     imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxx',
//     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:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxx',
//     imageSetState: 'LOCKED',
//     imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//     latestVersionId: '4',
//     updatedAt: 2023-09-27T19:46:21.824Z
//   }
// }
return response;
```

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

コピー先を指定せずにイメージセットをコピーします。

```
await copyImageSet(  
  "12345678901234567890123456789012",  
  "12345678901234567890123456789012",  
  "1",  
);
```

コピー先を指定してイメージセットをコピーします。

```
await copyImageSet(  
  "12345678901234567890123456789012",  
  "12345678901234567890123456789012",  
  "1",  
  "12345678901234567890123456789012",  
  "1",  
  false,  
);
```

イメージセットのサブセットを送信先にコピーし、コピーを強制します。

```
await copyImageSet(  
  "12345678901234567890123456789012",  
  "12345678901234567890123456789012",  
  "1",  
  "12345678901234567890123456789012",  
  "1",  
  true,  
  ["12345678901234567890123456789012", "11223344556677889900112233445566"],  
);
```

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

 Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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,
        force=False,
        subsets=[],
    ):
        """
        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.
        :param force: Force the copy.
        :param subsets: The optional subsets to copy. For example:
["12345678901234567890123456789012"].
        :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,
            }
        if len(subsets) > 0:
            copySubsetsJson = {
                "SchemaVersion": "1.1",
                "Study": {"Series": {"imageSetId": {"Instances": {}}}},
            }

            for subset in subsets:
                copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[
                    subset
                ] = {}

            copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
                "copiableAttributes": json.dumps(copySubsetsJson)
            }
        copy_results = self.health_imaging_client.copy_image_set(
            datastoreId=datastore_id,
            sourceImageSetId=image_set_id,
            copyImageSetInformation=copy_image_set_information,
            force=force,
        )
    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,
    force=force,
)
```

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,
    force=force,
)
```

イメージセットのサブセットをコピーします。

```
copy_image_set_information = {
```

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

    if len(subsets) > 0:
        copySubsetsJson = {
            "SchemaVersion": "1.1",
            "Study": {"Series": {"imageSetId": {"Instances": {}}}},
        }

        for subset in subsets:
            copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[
                subset
            ] = {}

        copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
            "copiableAttributes": json.dumps(copySubsetsJson)
        }

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

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「[CopyImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_source_image_set_id = '1234567890123456789012345678901234567890'
  " iv_source_version_id = '1'
  " iv_destination_image_set_id =
'1234567890123456789012345678901234567890' (optional)
  " iv_destination_version_id = '1' (optional)
  " iv_force = abap_false
  DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
    iv_latestversionid = iv_source_version_id ).
  DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(
    io_sourceimageset = lo_source_info ).
  IF iv_destination_image_set_id IS NOT INITIAL AND
    iv_destination_version_id IS NOT INITIAL.
    DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
      iv_imagesetid = iv_destination_image_set_id
      iv_latestversionid = iv_destination_version_id ).
    lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
      io_sourceimageset = lo_source_info
      io_destinationimageset = lo_dest_info ).
  ENDIF.
  oo_result = lo_mig->copyimageset(
    iv_datastoreid = iv_datastore_id
    iv_sourceimagesetid = iv_source_image_set_id
    io_copyimagesetinformatoin = lo_copy_info
    iv_force = iv_force ).
  DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
  DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
  MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
  MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
```

```

MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.

```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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

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

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 getopt 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 getopt "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、AWS CLI コマンドリファレンスの「[CreateDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

CLI

AWS CLI

例 1：データストアにタグを付けるには

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。を指定せずにデータストアを作成する場合 --lossless-storage-format、AWS HealthImaging のデフォルトは HTJ2K (高スループット JPEG 2000) です。

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

出力:

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

例 2: JPEG 2000 ロスレスストレージ形式でデータストアを作成するには

JPEG 2000 ロスレスストレージ形式で設定されたデータストアは、可逆イメージフレームを JPEG 2000 形式でトランスコードして保持します。その後、画像フレームは JPEG 2000 ロスレス、トランスコーディングなしで取得できます。次の create-datastore コード例では、my-datastore という名前の JPEG 2000 ロスレスストレージ形式用に設定されたデータストアを作成します。

```
aws medical-imaging create-datastore \
  --datastore-name "my-datastore" \
  --lossless-storage-format JPEG_2000_LOSSLESS
```

出力:

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

詳細については、「AWS HealthImaging Developer Guide」の「[Creating a data store](#)」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「[CreateDatastore](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[CreateDatastore](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',  
//    datastoreStatus: 'CREATING'  
// }  
return response;  
};
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[CreateDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
    " iv_datastore_name = 'my-datastore-name'
    oo_result = lo_mig->createdatastore( iv_datastorename =
iv_datastore_name ).
    DATA(lv_datastore_id) = oo_result->get_datastoreid( ).
    MESSAGE 'Data store created.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict. Data store may already exist.' TYPE 'I'.
```

```
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migservicequotaexcdex.  
  MESSAGE 'Service quota exceeded.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください [AWS SDK でのこのサービスの使用](#)。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` を使用する

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

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 getopt 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 getopt "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
}
```

```
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、「AWS CLI コマンドリファレンス」の「[DeleteDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Deleting a data store](#)」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「[DeleteDatastore](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[DeleteDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'DELETING'
  // }

  return response;
};
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[DeleteDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->deletedatastore( iv_datastoreid = iv_datastore_id ).  
  MESSAGE 'Data store deleted.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict. Data store may contain resources.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Data store not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[DeleteDatastore](#)を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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 を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [画像セットと画像フレームを使い始める](#)

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、AWS SDK for C++ API リファレンスの「[DeleteImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Deleting an image set](#)」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「[DeleteImageSet](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[DeleteImageSet](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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 = "xxxxxxxxxxxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxxxxxxxxxx",
) => {
    const response = await medicalImagingClient.send(
```

```
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   datastoreId: 'xxxxxxxxxxxxxxxxxx',
//   imageSetId: 'xxxxxxxxxxxxxxxxxx',
//   imageSetState: 'LOCKED',
//   imageSetWorkflowStatus: 'DELETING'
// }
return response;
};
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[DeleteImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  " iv_image_set_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->deleteimageset(  
    iv_datastoreid = iv_datastore_id  
    iv_imagesetid = iv_image_set_id ).  
  MESSAGE 'Image set deleted.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Image set not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[DeleteImageSet](#)を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDKでのこのサービスの使用](#)。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` を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [画像セットと画像フレームを使い始める](#)

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、AWS SDK for C++ API リファレンスの「[GetDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Getting import job properties](#)」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「[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、AWS SDK for Java 2.x API リファレンスの「[GetDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 = "xxxxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxxxx",
) => {
  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/
  output_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxx'/',
  //     submittedAt: 2023-09-19T17:27:25.143Z
  //   }
  // }

  return response;
};
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Python (Boto3) API リファレンスの「[GetDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_job_id = '12345678901234567890123456789012'
  oo_result = lo_mig->getdicomimportjob(
    iv_datastoreid = iv_datastore_id
    iv_jobid = iv_job_id ).
  DATA(lo_job_props) = oo_result->get_jobproperties( ).
  DATA(lv_job_status) = lo_job_props->get_jobstatus( ).
  MESSAGE |Job status: { lv_job_status }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Job not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
```

```
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[GetDICOMImportJob](#)を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDKでのこのサービスの使用](#)。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

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

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 getopt 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 getopt "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、「AWS CLI コマンドリファレンス」の「[GetDatastore](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

CLI

AWS CLI

例 1: 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",
    "losslessStorageFormat": "HTJ2K"
    "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"
  }
}
```

例 2: JPEG 2000 用に設定された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、JPEG 2000 ロスレスストレージ形式用に設定されたデータストアのプロパティを取得します。

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

出力:

```
{
  "datastoreProperties": {
    "datastoreId": "12345678901234567890123456789012",
    "datastoreName": "TestDatastore123",
    "datastoreStatus": "ACTIVE",
    "losslessStorageFormat": "JPEG_2000_LOSSLESS",
    "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 Developer Guide」の「[Getting data store properties](#)」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「[GetDatastore](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[GetDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreName: 'my_datastore',
  //     datastoreStatus: 'ACTIVE',
  //     updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response.datastoreProperties;
};
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[GetDatastore](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->getdatastore( iv_datastoreid = iv_datastore_id ).  
  DATA(lo_properties) = oo_result->get_datastoreproperties( ).  
  DATA(lv_name) = lo_properties->get_datastorename( ).  
  DATA(lv_status) = lo_properties->get_datastorestatus( ).  
  MESSAGE 'Data store properties retrieved.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Data store not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの[GetDatastore](#)を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [画像セットと画像フレームを使い始める](#)

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、AWS SDK for C++ API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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.jpg
```

注：このコード例には出力は含まれていません。GetImageFrame という操作は、ピクセルデータのストリームを imageframe.jpg ファイルに返すからです。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 Developer Guide」の「[Getting image set pixel data](#)」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「[GetImageFrame](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for JavaScript API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[GetImageFrame](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  " iv_image_set_id = '1234567890123456789012345678901234567890'  
  " iv_image_frame_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->getimageframe(  
    iv_datastoreid = iv_datastore_id  
    iv_imagesetid = iv_image_set_id  
    io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(  
      iv_imageframeid = iv_image_frame_id ) ).  
  DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).  
  MESSAGE 'Image frame retrieved.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Image frame not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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

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

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 Developer Guide」の「[Getting image set properties](#)」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「[GetImageSet](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[GetImageSet](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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.
```


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

 Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[GetImageSet](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_image_set_id = '1234567890123456789012345678901234567890'
  " iv_version_id = '1' (optional)
  IF iv_version_id IS NOT INITIAL.
    oo_result = lo_mig->getimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id
      iv_versionid = iv_version_id ).
  ELSE.
    oo_result = lo_mig->getimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
  ENDIF.
```

```
DATA(lv_state) = oo_result->get_imagesetstate( ).
MESSAGE |Image set retrieved with state: { lv_state }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcefoundex.
MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[GetImageSet](#)を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDKでのこのサービスの使用](#)。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`で使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [画像セットと画像フレームを使い始める](#)

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 outputPath: 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、AWS SDK for C++ API リファレンスの「[GetImageSetMetadata](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Getting image set metadata](#)」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「[GetImageSetMetadata](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[GetImageSetMetadata](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 "node: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.gz",
  datastoreId = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  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、AWS SDK for JavaScript API リファレンスの「[GetImageSetMetadata](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[GetImageSetMetadata](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_image_set_id = '1234567890123456789012345678901234567890'
  " iv_version_id = '1' (optional)
  IF iv_version_id IS NOT INITIAL.
    oo_result = lo_mig->getimagesetmetadata(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id
      iv_versionid = iv_version_id ).
  ELSE.
    oo_result = lo_mig->getimagesetmetadata(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
  ENDIF.
  DATA(lv_metadata_blob) = oo_result->get_imagesetmetadatablob( ).
  MESSAGE 'Image set metadata retrieved.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
```

```
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Image set not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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

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

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 Developer Guide」の「[Listing import jobs](#)」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、「AWS SDK for Java 2.x API リファレンス」の「[ListDICOMImportJobs](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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 = "xxxxxxxxxxxxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

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

  const 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       endedAt: 2023-09-22T14:49:51.351Z,
//       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       jobName: 'test-1',
//       jobStatus: 'COMPLETED',
//       submittedAt: 2023-09-22T14:48:45.767Z
//     }
//   ]
}

return jobSummaries;
};

```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_datastore_id = '1234567890123456789012345678901234567890'  
    oo_result = lo_mig->listdicomimportjobs( iv_datastoreid =  
iv_datastore_id ).  
    DATA(lt_jobs) = oo_result->get_jobsummaries( ).  
    DATA(lv_count) = lines( lt_jobs ).  
    MESSAGE |Found { lv_count } DICOM import jobs.| TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
    MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Resource not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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` で使用する

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

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 getopt 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 getopt "h" option; do
        case "${option}" in
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
        esac
    done
}
```

```
        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、AWS CLI コマンドリファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Listing data stores](#)」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「[ListDatastores](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       datastoreName: 'my_datastore',
//       datastoreStatus: 'ACTIVE',
//       updatedAt: 2023-08-04T18:49:54.429Z
//     }
//     ...
//   ]
// }

return datastoreSummaries;
};
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[ListDatastores](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    oo_result = lo_mig->listdatastores( ).  
    DATA(lt_datastores) = oo_result->get_datastoresummaries( ).  
    DATA(lv_count) = lines( lt_datastores ).  
    MESSAGE |Found { lv_count } data stores.| TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[ListDatastores](#)」を参照してください。

 Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDKでのこのサービスの使用](#)。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**で使用する

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

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 Developer Guide」の「[Listing image set versions](#)」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「[ListImageSetVersions](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[ListImageSetVersions](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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,
  );

  const 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: 'xxxxxxxxxxxxxxxxxxxxxxxx',
//       imageSetState: 'ACTIVE',
//       versionId: '1'
//     }
//   ]
// }
return imageSetPropertiesList;
};
```

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[ListImageSetVersions](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  " iv_image_set_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->listimagesetversions(  
    iv_datastoreid = iv_datastore_id  
    iv_imagesetid = iv_image_set_id ).  
  DATA(lt_versions) = oo_result->get_imagesetpropertieslist( ).  
  DATA(lv_count) = lines( lt_versions ).  
  MESSAGE |Found { lv_count } image set versions.| TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Image set not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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` で使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [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/1234567890123456789012/  
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「HealthImaging を使用した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、「AWS CLI コマンドリファレンス」の「[ListTagsForResource](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[ListTagsForResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、「AWS SDK for JavaScript API リファレンス」の「[ListTagsForResource](#)」を参照してください。

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[ListTagsForResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
    " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
    oo_result = lo_mig->listtagsforresource( iv_resourcearn =
iv_resource_arn ).
    DATA(lt_tags) = oo_result->get_tags( ).
    DATA(lv_count) = lines( lt_tags ).
    MESSAGE |Found { lv_count } tags for resource.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcefoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[ListTagsForResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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` を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [画像セットと画像フレームを使い始める](#)

C++

SDK for C++

画像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(patientID)});

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: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```

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

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

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

useCase2EndDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAndTime()
.WithDICOMStudyDate(Aws::Utils::DateTime(std::chrono::system_clock::now()).ToLocalTimeStr("%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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して、updatedAt フィールドで ASC 順序にレスポンスをソートします。

```

    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、AWS SDK for C++ API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 : DICOMStudyDate と DICOMStudyTime を使用することで、BETWEEN 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、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 : DICOMStudyTime は選択可能です。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 Huel1628",
        "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の間で、HealthImaging に保持されている DICOM スタディを含む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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して画像セットを検索し、updatedAt フィールドのレスポンスを ASC 順序でソートするには

次の search-image-sets コード例では、DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して画像セットを検索し、updatedAt フィールドのレスポンスを ASC 順序でソートします。

注: updatedAt 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": [{
      "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 Developer Guide」の「[Searching image sets](#)」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「[SearchImageSets](#)」を参照してください。

Java

SDK for Java 2.x

画像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: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```
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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して、updatedAt フィールドで ASC 順序にレスポンスをソートします。

```
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、AWS SDK for Java 2.x API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

画像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: [{ DICOPatientId: "1234567" }],
        operator: "EQUAL",
      },
    ],
  };
};
```

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

ユースケース #2: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```
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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して、updatedAt フィールドで ASC 順序にレスポンスをソートします。

```
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",
          },
        ],
      },
    ],
  };

  await searchImageSets(datastoreId, searchCriteria);
}
```

```
        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、AWS SDK for JavaScript API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

画像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: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```

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: DICOMSeriesInstanceUID で EQUAL 演算子を使用し、updatedAt で BETWEEN 演算子を使用して、updatedAt フィールドで ASC 順序にレスポンスをソートします。

```
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、AWS SDK for Python (Boto3) API リファレンスの「[SearchImageSets](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
  " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
  oo_result = lo_mig->searchimagesets(
    iv_datastoreid = iv_datastore_id
    io_searchcriteria = io_search_criteria ).
  DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).
  DATA(lv_count) = lines( lt_imagesets ).
  MESSAGE |Found { lv_count } image sets.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
```

```
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [画像セットと画像フレームを使い始める](#)

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、AWS SDK for C++ API リファレンスの「[StartDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Developer Guide」の「[Starting an import job](#)」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、AWS SDK for Java 2.x API リファレンスの「[StartDICOMImportJob](#)」を参照してください。

 Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',

```

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

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

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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 Python (Boto3) API リファレンスの「[StartDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '1234567890123456789012345678901234567890'
```

```
" iv_role_arn = 'arn:aws:iam::123456789012:role/ImportJobRole'
" iv_input_s3_uri = 's3://my-bucket/input/'
" iv_output_s3_uri = 's3://my-bucket/output/'
oo_result = lo_mig->startdicomimportjob(
  iv_jobname = iv_job_name
  iv_datastoreid = iv_datastore_id
  iv_dataaccessrolearn = iv_role_arn
  iv_inputs3uri = iv_input_s3_uri
  iv_outputs3uri = iv_output_s3_uri ).
DATA(lv_job_id) = oo_result->get_jobid( ).
MESSAGE |DICOM import job started with ID: { lv_job_id }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
  MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[StartDICOMImportJob](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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

次のサンプルコードは、`TagResource` を使用方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [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/  
image-set/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「HealthImaging を使用したリソースのタグ付け」](#)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「[TagResource](#)」を参照してください。

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、AWS SDK for Java 2.x API リファレンスの「[TagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for JavaScript API リファレンスの「[TagResource](#)」を参照してください。

Note

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[TagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_resource_arn = 'arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012'  
    lo_mig->tagresource(  
        iv_resourcearn = iv_resource_arn  
        it_tags = it_tags ).  
    MESSAGE 'Resource tagged successfully.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Resource not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの[TagResource](#)を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [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/1234567890123456789012/  
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「HealthImaging を使用した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、「AWS CLI Command Reference」の「[UntagResource](#)」を参照してください。

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、「AWS SDK for Java 2.x API リファレンス」の「[UntagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、「AWS SDK for JavaScript API リファレンス」の「[UntagResource](#)」を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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、AWS SDK for Python (Boto3) API リファレンスの「[UntagResource](#)」を参照してください。

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_resource_arn = 'arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012'  
    lo_mig->untagresource(  
        iv_resourcearn = iv_resource_arn  
        it_tagkeys = it_tag_keys ).  
    MESSAGE 'Resource untagged successfully.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Resource not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP [UntagResource](#)」を参照してください。

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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` を使用する

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

CLI

AWS CLI

例 1: 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` の例では、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 \
  --cli-binary-format raw-in-base64-out \
  --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json` の内容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{\"PatientName\": \"MX^MX\"}}}"
  }
}
```

出力:

```
{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
  "updatedAt": 1680042257.908,
  "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
  "imageSetState": "LOCKED",
```

```
"createdAt": 1680027126.436,  
"datastoreId": "12345678901234567890123456789012"  
}
```

例 2: 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` の例では、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 \  
  --cli-binary-format raw-in-base64-out \  
  --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json` の内容

```
{  
  "DICOMUpdates": {  
    "removableAttributes": "{\"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"  
}
```

例 3: 画像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` の例では、画像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 \  
  --cli-binary-format raw-in-base64-out \  
  --update-image-set-metadata-updates file://metadata-updates.json \  
  --force
```

metadata-updates.json の内容

```
{  
  "DICOMUpdates": {  
    "removableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series  
  \": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\":  
    {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}\"  
  }  
}
```

出力:

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

例 4: 画像セットを以前のバージョンに戻すには

次の update-image-set-metadata の例は、画像セットを以前のバージョンに戻す方法を示しています。CopyImageSet および UpdateImageSetMetadata アクションは、新しいバージョンの画像セットを作成します。

```
aws medical-imaging update-image-set-metadata \  
  --datastore-id 12345678901234567890123456789012 \  
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \  
  --latest-version-id 3 \  
  --cli-binary-format raw-in-base64-out \  
  --force
```

```
--update-image-set-metadata-updates '{"revertToVersionId": "1"}'
```

出力:

```
{
  "datastoreId": "12345678901234567890123456789012",
  "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
  "latestVersionId": "4",
  "imageSetState": "LOCKED",
  "imageSetWorkflowStatus": "UPDATING",
  "createdAt": 1680027126.436,
  "updatedAt": 1680042257.908
}
```

例 5: インスタンスにプライベート DICOM データ要素を追加するには

次の update-image-set-metadata の例は、画像セット内で指定されたインスタンスにプライベート要素を追加する方法を示しています。DICOM 標準では、標準データ要素に含めることができない情報の通信に、プライベートデータ要素が許可されます。UpdateImageSetMetadata アクションを使用して、プライベートデータ要素を作成、更新、削除できます。

```
aws medical-imaging update-image-set-metadata \
  --datastore-id 12345678901234567890123456789012 \
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
  --latest-version-id 1 \
  --cli-binary-format raw-in-base64-out \
  --force \
  --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json の内容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\": {\"001910F9\": \"97\"}, \"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}}"
  }
}
```

出力:

```
{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
  "updatedAt": 1680042257.908,
  "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
  "imageSetState": "LOCKED",
  "createdAt": 1680027126.436,
  "datastoreId": "12345678901234567890123456789012"
}
```

例 6: プライベート DICOM データ要素をインスタンスに対して更新するには

次の `update-image-set-metadata` の例は、画像セット内のインスタンスに属するプライベートデータ要素の値を更新する方法を示しています。

```
aws medical-imaging update-image-set-metadata \
  --datastore-id 12345678901234567890123456789012 \
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
  --latest-version-id 1 \
  --cli-binary-format raw-in-base64-out \
  --force \
  --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json` の内容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\": {\"00091001\": \"GE_GENESIS_DD\"}}}}}}}"
  }
}
```

出力:

```
{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
```

```

    "updatedAt": 1680042257.908,
    "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
  }

```

例 7: force パラメータを使用して SOPInstanceUID を更新するには

次の update-image-set-metadata の例は、force パラメータを使用して SOPInstanceUID を更新し、DICOM メタデータの制約をオーバーライドする方法を示しています。

```

aws medical-imaging update-image-set-metadata \
  --datastore-id 12345678901234567890123456789012 \
  --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
  --latest-version-id 1 \
  --cli-binary-format raw-in-base64-out \
  --force \
  --update-image-set-metadata-updates file://metadata-updates.json

```

metadata-updates.json の内容

```

{
  "DICOMUpdates": {
    "updatableAttributes": "{\\"SchemaVersion\\":1.1,\\"Study\\":{\\"Series
\\":{\\"1.3.6.1.4.1.5962.99.1.3633258862.2104868982.1369432891697.3656.0\\":
{\\"Instances\\":
{\\"1.3.6.1.4.1.5962.99.1.3633258862.2104868982.1369432891697.3659.0\\":{\\"DICOM\\":
{\\"SOPInstanceUID\\":
\\"1.3.6.1.4.1.5962.99.1.3633258862.2104868982.1369432891697.3659.9\\"}}}}}}}"
  }
}

```

出力:

```

{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
  "updatedAt": 1680042257.908,
  "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
  "imageSetState": "LOCKED",
  "createdAt": 1680027126.436,

```

```
"datastoreId": "12345678901234567890123456789012"  
}
```

詳細については、「AWS HealthImaging Developer Guide」の「[Updating image set metadata](#)」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「[UpdateImageSetMetadata](#)」を参照してください。

Java

SDK for Java 2.x

```
/**  
 * Update the metadata of an AWS HealthImaging image set.  
 *  
 * @param medicalImagingClient - The AWS HealthImaging client object.  
 * @param datastoreId          - The datastore ID.  
 * @param imageSetId          - The image set ID.  
 * @param versionId           - The version ID.  
 * @param metadataUpdates     - A MetadataUpdates object containing the  
updates.  
 * @param force                - The force flag.  
 * @throws MedicalImagingException - Base exception for all service  
exceptions thrown by AWS HealthImaging.  
 */  
public static void updateMedicalImageSetMetadata(MedicalImagingClient  
medicalImagingClient,  
                                                String datastoreId,  
                                                String imageSetId,  
                                                String versionId,  
                                                MetadataUpdates  
metadataUpdates,  
                                                boolean force) {  
    try {  
        UpdateImageSetMetadataRequest updateImageSetMetadataRequest =  
UpdateImageSetMetadataRequest  
            .builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imageSetId)  
            .latestVersionId(versionId)  
            .updateImageSetMetadataUpdates(metadataUpdates)
```

```
        .force(force)
        .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());
        throw e;
    }
}
```

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, force);
```

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, force);

```

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, force);

```

ユースケース 4: 以前のバージョンに戻します。

```

        // In this case, revert to previous version.
        String revertVersionId =
Integer.toString(Integer.parseInt(versionid) - 1);
        MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
            .revertToVersionId(revertVersionId)
            .build();
        updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
            versionid, metadataRemoveUpdates, force);

```

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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.
* @param {boolean} force - Force the update.
*/
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   imageSetState: 'LOCKED',
    //   imageSetWorkflowStatus: 'UPDATING',
    //   latestVersionId: '4',
    //   updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
  } catch (err) {
```

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

ユースケース 1: 属性を挿入または更新し、強制的に更新します。

```
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,
  true,
);
```

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,
);
```

ユースケース 4: 以前のバージョンに戻します。

```
const updateMetadata = {
  revertToVersionId: "1",
};

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

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

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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, force=False
    ):
        """
        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\"}}}}"}
        :param: force: Force the update.
        :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,
                force=force,
            )
        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, force
)
```

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, force
)
```

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, force  
)
```

ユースケース 4: 以前のバージョンに戻します。

```
metadata = {"revertToVersionId": "1"}  
  
self.update_image_set_metadata(  
    data_store_id, image_set_id, version_id, metadata, force  
)
```

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

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

SAP ABAP

SDK for SAP ABAP

```
TRY.  
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'  
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'  
    " iv_latest_version_id = '1'  
    " iv_force = abap_false  
    oo_result = lo_mig->updateimagesetmetadata(  
        iv_datastoreid = iv_datastore_id  
        iv_imagesetid = iv_image_set_id  
        iv_latestversionid = iv_latest_version_id  
        io_updateimagesetmetupdates = io_metadata_updates  
        iv_force = iv_force ).  
    DATA(lv_new_version) = oo_result->get_latestversionid( ).
```

```
MESSAGE |Image set metadata updated to version: { lv_new_version }.| TYPE
'I'.
CATCH /aws1/cx_migaccessdeniedex.
MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの[UpdateImageSetMetadata](#)を参照してください。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した HealthImaging のシナリオ AWS SDKs

次のコード例は、AWS SDKs を使用した HealthImaging で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、HealthImaging 内で複数の関数を呼び出すか、他の AWS のサービスと組み合わせて、特定のタスクを実行する方法を示しています。各シナリオには、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

シナリオは、サービスアクションをコンテキストで理解するのに役立つ中級レベルの経験を対象としています。

例

- [AWS SDK を使用して HealthImaging イメージセットとイメージ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 イメージセットのタグ付け](#)

AWS SDK を使用して HealthImaging イメージセットとイメージフレームの使用を開始する

次のコード例は、HealthImaging で DICOM ファイルをインポートし、画像フレームをダウンロードする方法を示しています。

実装は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 インポート用にリソースをセットアップします。
- 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 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,
                                                         const
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 outputPath: 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();
}
```

画像フレームをダウンロード、デコード、および検証します。

```
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
signed bytes - "
            << "verifyChecksumForImage, unsupported data type,
            << 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
unsigned bytes - "
                << "verifyChecksumForImage, unsupported data type,
                << 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

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

ステップをオーケストレーションします (index.js)。

```
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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
    parseScenarioArgs(scenarios, {
        name: "Health Imaging Workflow",
        description:
            "Work with DICOM images using an AWS Health Imaging data store.",
    });
}
```

```
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes]
[-v|--verbose]",
    });
  }
}
```

リソースをデプロイします (deploy-steps.js)。

```
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,
  "../../../../../scenarios/features/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 },
);
```

DICOM ファイルをコピーします (dataset-steps.js)。

```
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.`
);
```

データストアへのインポートを開始します (import-steps.js)。

```
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",
    default: true,
  },
);

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}`,
);
```

画像セット ID を取得します (image-set-steps.js -)。

```
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((ids, next) => {
        return Object.assign({}, ids, {
          [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")}` ,
  );
```

画像フレーム ID を取得します (image-frame-steps.js)。

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node: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`;\n    }\n\n    return output;\n  },\n);
```

画像フレームを確認します (verify-steps.js)。 [AWS HealthImaging Pixel データ検証ライブラリ](#) が検証に使用されました。

```
import { spawn } from "node:child_process";\n\nimport {\n  ScenarioAction,\n  ScenarioInput,\n} from "@aws-doc-sdk-examples/lib/scenario/index.js";\n\n/**\n * @typedef {Object} DICOMValueRepresentation\n * @property {string} name\n * @property {string} type\n * @property {string} value\n */\n\n/**\n * @typedef {Object} ImageFrameInformation\n * @property {string} ID\n * @property {Array<{ Checksum: number, Height: number, Width: number }>}\n  PixelDataChecksumFromBaseToFullResolution\n * @property {number} MinPixelValue\n * @property {number} MaxPixelValue\n * @property {number} FrameSizeInBytes\n */\n\n/**\n * @typedef {Object} DICOMMetadata\n * @property {Object} DICOM\n * @property {DICOMValueRepresentation[]} DICOMVRs\n * @property {ImageFrameInformation[]} ImageFrames\n */
```

```
/**
 * @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",
    default: true,
  },
);

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 [seriesInstanceId, series] of Object.entries(
      metadata.Study.Series,
    )) {
      for (const [sopInstanceId, _] of Object.entries(series.Instances)) {
        console.log(
          `Verifying image set ${imageSetId} with series ${seriesInstanceId}
and sop ${sopInstanceId}`,
        );
        const child = spawn(
          "node",
          [
            verificationTool,
            datastoreId,
            imageSetId,
            seriesInstanceId,
            sopInstanceId,
          ],
          { 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}`,
                ),
              );
            }
          });
        });
      }
    }
  });
}
```

```
    });  
  }  
}  
},  
);
```

リソースを破壊します (clean-up-steps.js)。

```
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

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

必要なリソースを持つ 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(
    "../../../../../scenarios/features/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 AWS 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 AWS 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 AWS 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
```

画像フレームをダウンロード、デコード、および検証します。

```
class MedicalImagingWrapper:
    """Encapsulates AWS 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\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\t\tImage parameters for {jph_file}: \n\t\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 opout in stack.outputs:
        if opout["OutputKey"] == "DatastoreID":
            data_store_id = opout["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 AWS 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

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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るには

```
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

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

JavaScript

SDK for JavaScript (v3)

データストアにタグを付けるには

```
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

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Python

SDK for Python (Boto3)

データストアにタグを付けるには

```
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

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください [AWS SDK でのこのサービスの使用](#)。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 イメージセットのタグ付け

次のコード例は、HealthImaging イメージ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

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWSコード例リポジトリ](#)での設定と実行の方法を確認してください。

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

GitHubには、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。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 HealthImaging のモニタリング

モニタリングとログ記録は、AWS HealthImaging 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。には、HealthImaging を監視し、問題が発生したときに報告し、必要に応じて自動アクションを実行するための以下のログ記録とモニタリングツール AWS が用意されています。

- 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。呼び出し元のユーザーとアカウント AWS、呼び出し元の送信元 IP アドレス、呼び出しの発生日時を特定できます。詳細については、「[AWS CloudTrail ユーザーガイド](#)」を参照してください。
- Amazon CloudWatch は、AWS リソースと で実行され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、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、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 CloudWatch ユーザーガイド](#)」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イベントソースのデータに簡単に接続できるようにするサーバーレスイベントバスサービスです。EventBridge は、お客様独自のアプリケーション、Software as a Service (SaaS) アプリケーション、AWS のサービスからのリアルタイムデータをストリーム配信し、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 EventBridge ユーザーガイド](#)」を参照してください。

トピック

- [HealthImaging AWS CloudTrail での の使用](#)
- [HealthImaging での Amazon CloudWatch の使用](#)
- [HealthImaging での Amazon EventBridge の使用](#)

HealthImaging AWS CloudTrail での の使用

AWS HealthImaging は AWS CloudTrail、HealthImaging のユーザー、ロール、または のサービスによって実行されたアクションを記録する AWS サービスであると統合されています。CloudTrail は、HealthImaging のすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、HealthImaging コンソールからのコールと、HealthImaging API オペレーションへのコードコールが含まれます。証跡を作成する場合は、HealthImaging のイベントを含む Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、HealthImaging に対するリクエスト、リクエスト元の IP アドレス、リクエスト元のユーザー、リクエスト日時などの詳細を確認できます。

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

証跡の作成

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

Note

で AWS HealthImaging の CloudTrail イベント履歴を表示するには AWS マネジメントコンソール、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`。

HealthImaging のイベントなど AWS アカウント、 のイベントの継続的な記録については、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。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さらに分析して処理するように他の AWS サービスを設定できます。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 がサポートされているサービスと統合](#)
- 「[CloudTrail の Amazon SNS 通知の設定](#)」
- [複数のリージョンから CloudTrail ログファイルを受け取る](#) および [複数のアカウントから CloudTrail ログファイルを受け取る](#)

Note

AWS HealthImaging は、管理イベントとデータイベントの 2 種類の CloudTrail イベントをサポートしています。管理イベントは、HealthImaging など、すべての AWS サービスが生成する一般的なイベントです。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。データイベントは請求可能で、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 ログをオプトアウトできます。

HealthImaging では、[AWS HealthImaging API リファレンスに記載されているすべてのネイティブ API](#) アクションは、を除いて管理イベントとして分類されま
す [GetImageFrame](#)。GetImageFrame アクションはデータイベントとして CloudTrail で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。

DICOMweb WADO-RS API アクションは CloudTrail のデータイベントとして分類されるため、オプトイン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 [HealthImaging から DICOM データを取得する](#) 「」および「[データイベントのログ記録](#)」を参照してください。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストがルートまたは AWS Identity and Access Management (IAM) ユーザー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

「トレイル」は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは、任意の出典からの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、GetDICOMImportJob アクションを示す HealthImaging のログエントリです。

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "XXXXXXXXXXXXXXXXXXXX: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": "XXXXXXXXXXXXXXXXXXXX",
        "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"
}

```

HealthImaging での Amazon CloudWatch の使用

CloudWatch を使用して AWS HealthImaging をモニタリングできます。CloudWatch は生データを収集し、それを読み取り可能なほぼリアルタイムのメトリクスに処理します。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の確に把握できます。また、特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「[Amazon CloudWatch ユーザーガイド](#)」を参照してください。

Note

メトリクスはすべての HealthImaging API について報告されます。

以下の表は、HealthImaging のメトリクスとディメンションの一覧です。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

メトリクス	説明
	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

HealthImaging のメトリクスは、AWS マネジメントコンソール、AWS CLI または CloudWatch API で取得できます。CloudWatch API は、いずれかの Amazon AWS Software Development Kit (SDK) または Amazon CloudWatch API ツールでも使用できます。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。

CloudWatch で HealthImaging をモニタリングするには、適切な CloudWatch アクセス権限が必要です。詳細については、「Amazon Aurora ユーザーガイド」の「[CloudWatch のアイデンティティとアクセス管理](#)」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方法 (CloudWatch コンソール)

1. にサインイン AWS マネジメントコンソールし、[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。

CloudWatch を使用したアラームの作成

CloudWatch アラームは指定期間中に単一のメトリクスを監視し、1 つ以上のアクションを実行して 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します。アクションは、複数の指定期間にわたって特定のしきい値を基準としたメトリクスの値に応じて実行されます。アラームの状態が変わったときにも、CloudWatch は Amazon SNS メッセージを送信できます。

CloudWatch アラームがアクションを呼び出すのは、状態が変わってから指定期間が経過するまで、その新しい状態が続いた場合に限りです。詳細については、「[CloudWatch アラームの使用](#)」を参照してください。

HealthImaging での Amazon EventBridge の使用

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同士を接続す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。EventBridge の基礎は、[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EventBridge に状態変更を永続的に配信します。詳細については、「Amazon EventBridge ユーザーガイド」の「[Amazon EventBridge とは](#)」を参照してください。

トピック

- [EventBridge に送信される HealthImaging イベント EventBridge](#)
- [HealthImaging イベント構造と例](#)

EventBridge に送信される HealthImaging イベント EventBridge

次の表に、処理のために EventBridge 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	State
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
データストアの削除	DELETED
詳細については、AWS HealthImaging 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
インポート中のジョブ	IN_PROGRESS

HealthImaging イベントタイプ	State
インポートジョブが完了しました	COMPLETED
ジョブのインポートに失敗しました	FAILED

詳細については、AWS HealthImaging API リファレンスの [jobStatus](#) を参照してください。
HealthImaging

イメージセットイベント	
イメージセットが作成されました	CREATED
イメージセットのコピー	COPYING
読み取り専用アクセスによるイメージセットのコピー	COPYING_WITH_READ_ONLY_ACCESS
コピーされたイメージセット	COPIED
イメージセットのコピーに失敗しました	COPY_FAILED
イメージセットの更新	UPDATING
イメージセットが更新されました	UPDATED
イメージセットの更新に失敗しました	UPDATE_FAILED
イメージセットの削除	DELETING
削除されたイメージセット	DELETED

詳細については、AWS HealthImaging API リファレンスの [ImageSetWorkflowStatus](#) 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に表に一覧表示されます。詳細について

は、「Amazon EventBridge ユーザーガイド」の「イベント[構造リファレンス](#)」を参照してください。

Note

HealthImaging イベント構造のsource属性は `aws.medical-imaging` です。

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。

名前	型	説明
id	string	イベントごとに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	HealthImaging の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/"
    }
  }
}

```

ジョブイベントのインポート - 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	イベントごとに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	HealthImaging の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。
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 バケットの入カ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 バケットの出カプレフィックス。

イメージセットイベント

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",
  "isPrimary": true,
  "imageSetVersion": "1",
  "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",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "COPYING",
  }
}
```

```

    "sourceImageSetArn": "arn:aws:medical-imaging:us-
west-2:147997158357:datastore/c381ee9b9ef34902a45b476dd7be068b/
imageset/0309de3674fd551fa7ddd2880b21f990"
  }
}

```

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",
    "isPrimary": true,
    "imageSetVersion": "1",
    "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",
  "isPrimary": true,
  "imageSetVersion": "1",
  "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",
    "isPrimary": true,
    "imageSetVersion": "1",
    "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",
  "isPrimary": true,
  "imageSetVersion": "1",
  "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",
    "isPrimary": true,
    "imageSetVersion": "1",
    "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",
    "isPrimary": true,
    "imageSetVersion": "1",
    "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",
```

```

    "isPrimary": true,
    "imageSetVersion": "1",
    "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",
    "isPrimary": true,
    "imageSetVersion": "1",
    "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	イベントごとに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)	イメージセットの 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	HealthImaging の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.isPrimary	boolean	インポートされたデータがマネージド階層に正常に整理されたかどうか、または解決する必要があるメタデータの競合があるかどうかを示します。

名前	型	説明
<code>detail.imageSetVersion</code>	string	イメージセットのバージョンは、インスタンスが複数回インポートされると増加します。最新バージョンは、プライマリイメージセットに保存されている古いバージョンを上書きします。
<code>detail.datastoreId</code>	string	ステータス変更イベントを生成したデータストア ID。
<code>detail.imagesetId</code>	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。
<code>detail.imageSetState</code>	string	現在のイメージセットの状態。
<code>detail.imageSetWorkflowStatus</code>	string	現在のイメージ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を活用できます。

セキュリティは、AWS お客様とお客様の間の責任共有です。[責任共有モデル](#)ではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- 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。は、お客様が安全に使用できるサービス AWS も提供します。サードパーティーの監査者は、[AWS コンプライアンスプログラム](#)コンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。が適用されるコンプライアンスプログラムの詳細については AWS 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、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、HealthImaging を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。次のトピックでは、セキュリティと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 サービスの使用方法についても説明します。

トピック

- [AWS HealthImaging でのデータ保護](#)
- [AWS HealthImaging のアイデンティティとアクセス管理](#)
- [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 HealthImaging でのデータ保護

責任 AWS [共有モデル](#)、AWS HealthImaging でのデータ保護に適用されます。このモデルで説明されているように、AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「AWS のサービス」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、[データプライバシーに関するよくある質問](#)を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された「[AWS 責任共有モデルおよび GDPR](#)」のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント、AWS IAM アイデンティティセンターまたは AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「AWS CloudTrail ユーザーガイド」の[CloudTrail 証跡の使用](#)を参照してください。
- AWS 暗号化ソリューションと、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 などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「[連邦情報処理規格 \(FIPS\) 140-3](#)」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して HealthImaging AWS CLI または他の AWS のサービスを使用する場合も同様です。AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

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 HealthImaging では、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でき、スケールで効率的な暗号化機能を提供しています。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時のデータ暗号化機能
- 暗号化キー AWS 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、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、AWS 環境で開発またはデプロイするサービスに統合するための APIs AWS を提供します。

保管中の暗号化

デフォルトでは、HealthImaging はサービス所有 AWS Key Management Service のキーを使用して保管中の顧客データを暗号化します。必要に応じて、作成、所有、管理する対称カスタマーマネージド AWS KMS キーを使用して、保管中のデータを暗号化するように HealthImaging を設定できます。詳細については、[「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。AWS Key Management Service

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経由で、またバックエンドサービス経由で、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 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 所有の 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、それら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 クォータには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

AWS KMS ライフサイクルと使用状況を完全に制御する場合、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 キーの使用をサポートします。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

CloudTrail を使用して、HealthImaging が AWS KMS ユーザーに代わって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 マネジメントコンソール 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 リソースでカスターマネージドキーを使用するには、キーポリシーで [kms:CreateGrant](#) オペレーションを許可する必要があります。これにより、指定した KMS キーへのアクセスを制御するカスターマネージドキーへの許可が付与され、ユーザーは HealthImaging で必要な [許可オペレーション](#) にアクセスできるようになります。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、そのキーで暗号化されたデータストアを、ユーザーが HealthImaging に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",
    "kms:GenerateDataKeyWithoutPlaintext"
  ],
  "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 暗号化を有効にして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、以下のポリシーに対するアクセス許可と、AWS HealthImaging に必要なアクセス許可が必要です。

HealthImaging が許可を使用する方法 AWS KMS

HealthImaging で、カスタマーマネージド KMS キーを使用するには[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、HealthImaging は [CreateGrant](#) リクエストを送信してユーザーに代わって許可を作成します AWS KMS。の許可 AWS KMS は、顧客アカウントの KMS キーへのアクセス権を HealthImaging に付与するために使用されます。

お客様の代わりに HealthImaging が作成する許可を廃止したり取り消したりしないでください。アカウントで AWS KMS キーを使用する許可を HealthImaging に付与する許可を取り消した、または廃止した場合、HealthImaging は、このデータへのアクセス、データストアにプッシュされた新しい画像リソースの暗号化、およびプルされた画像の復号化を行うことができなくなります。HealthImaging の許可を廃止または取り消すと、変更がすぐに適用されます。アクセス権限を取り消すには、許可を取り消すのではなく、データストアを削除します。データストアを削除すると、HealthImaging がユーザーに代わって許可を廃止します。

HealthImaging の暗号化キーのモニタリング

CloudTrail を使用して、カスタマーマネージド KMS キーを使用するときに HealthImaging が AWS KMS ユーザーに代わって送信するリクエストを追跡できます。CloudTrail ログのログエントリは、HealthImaging からのリクエストを明確に区別するため userAgent フィールドに `medical-imaging.amazonaws.com` を表示します。

以下はCreateGrant、カスタ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 HealthImaging 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するためのGenerateDataKey、Decrypt、、、およびの CloudTrail イベントの例です。

以下は、CreateGrant を使用して HealthImaging に顧客提供の KMS キーへのアクセスを許可し、HealthImaging がその KMS キーを使用して保管中のすべての顧客データを暗号化できるようにする方法を示しています。

ユーザーは自分で許可を作成する必要はありません。HealthImaging は、CreateGrantリクエストを送信することで、ユーザーに代わって許可を作成します AWS KMS。の許可 AWS KMS は、顧客アカウントの AWS KMS キーへのアクセスを HealthImaging に付与するために使用されます。

```
{
  "KeyId": "arn:aws:kms:us-east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
  "GrantId": "44e88bc45b769499ce5ec4abd5ecb27eeb3b178a4782452aae65fe885ee5ba20",
  "Name": "MedicalImagingGrantForQID0_ebfff634a-2d16-4046-9238-e3dc4ab54d29",
  "CreationDate": "2025-04-17T20:12:49+00:00",
  "GranteePrincipal": "AWS Internal",
  "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
  "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
  "Operations": [
    "Decrypt",
    "Encrypt",
    "GenerateDataKey",
    "GenerateDataKeyWithoutPlaintext",
    "ReEncryptFrom",
    "ReEncryptTo",
    "CreateGrant",
    "RetireGrant",
    "DescribeKey"
  ]
},
{
  "KeyId": "arn:aws:kms:us-east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
  "GrantId": "9e5fd5ba7812daf75be4a86efb2b1920d6c0c9c0b19781549556bf2ff98953a1",
  "Name": "2025-04-17T20:12:38",
  "CreationDate": "2025-04-17T20:12:38+00:00",
  "GranteePrincipal": "medical-imaging.us-east-1.amazonaws.com",
  "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
```

```

    "IssuingAccount": "AWS Internal",
    "Operations": [
      "Decrypt",
      "Encrypt",
      "GenerateDataKey",
      "GenerateDataKeyWithoutPlaintext",
      "ReEncryptFrom",
      "ReEncryptTo",
      "CreateGrant",
      "RetireGrant",
      "DescribeKey"
    ]
  },
  {
    "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
    "GrantId":
"ab4a9b919f6ca8eb2bd08ee72475658ee76cfc639f721c9caaa3a148941bcd16",
    "Name": "9d060e5b5d4144a895e9b24901088ca5",
    "CreationDate": "2025-04-17T20:12:39+00:00",
    "GranteePrincipal": "AWS Internal",
    "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
    "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
    "Operations": [
      "Decrypt",
      "Encrypt",
      "GenerateDataKey",
      "GenerateDataKeyWithoutPlaintext",
      "ReEncryptFrom",
      "ReEncryptTo",
      "DescribeKey"
    ],
    "Constraints": {
      "EncryptionContextSubset": {
        "kms-arn": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c"
      }
    }
  }
}

```

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、HealthImaging が Decrypt オペレーションを呼び出して、保管済みの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、HealthImaging が DescribeKey オペレーションを使用して、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"
}
```

詳細情報

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 とオンプレミスのアプリケーション間、および HealthImaging と Amazon S3 間の両方で保護されます。HealthImaging との間でのトラフィックは、デフォルトで HTTPS AWS Key Management Service を使用します。

- AWS HealthImaging はリージョンサービスで、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) リージョンで利用可能です。
- HealthImaging と Amazon S3 バケット間のトラフィックについては、Transport Layer Security (TLS) が HealthImaging と Amazon S3 間、および HealthImaging とそれにアクセスするカスタマーアプリケーション間で転送されるデータを暗号化します。Amazon S3 バケットの IAM ポリシーで [aws:SecureTransport condition](#) を使用して、HTTPS (TLS) 経由での暗号化された接続のみを許可するようにしてください。現在、HealthImaging では Amazon S3 バケット内のデータへのアクセスにパブリックエンドポイントを使用していますが、これにより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 HealthImaging のアイデンティティとアクセス管理

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、誰が認証 (サインイン) され、HealthImaging リソースを使用する認可を受ける (アクセス許可がある) ことができるかを管理します。IAM は、追加料金なしで使用できる AWS のサービスです。

トピック

- [オーディエンス](#)
- [アイデンティティを使用した認証](#)
- [ポリシーを使用したアクセスの管理](#)
- [AWS HealthImaging が IAM で機能する仕組み](#)
- [AWS HealthImaging のアイデンティティベースのポリシーの例](#)
- [AWS HealthImaging の マネージドポリシー](#)
- [HealthImaging でのサービス間の混乱した代理の防止](#)
- [AWS HealthImaging の ID とアクセスのトラブルシューティング](#)

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします ([「AWS HealthImaging の ID とアクセスのトラブルシューティング」](#)を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します ([「AWS HealthImaging が IAM で機能する仕組み」](#)を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します ([「AWS HealthImaging のアイデンティティベースのポリシーの例」](#)を参照)

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用してにサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーテッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の [「AWS アカウントにサインインする方法」](#)を参照してください。

プログラムによるアクセスの場合、は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の [「API リクエストに対するAWS 署名バージョン 4」](#)を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント ルートユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の [「ルートユーザー認証情報が必要なタスク」](#)を参照してください。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用してにアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーテッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID Directory Service ソースの認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[IAM アイデンティティセンターとは](#)」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、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「[ID プロバイダーとのフェデレーションを使用して にアクセスする必要がある AWS](#)」を参照してください。

[IAM グループ](#)は、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「[IAM ユーザーに関するユースケース](#)」を参照してください。

IAM ロール

[IAM ロール](#)は、特定のアクセス許可を持つアイデンティティであり、一時的な認証情報を提供します。[ユーザーから IAM ロール \(コンソール\) に切り替えるか、または API オペレーションを呼び出すことで、ロールを引き受けることができます。](#) AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「[ロールを引き受けるための各種方法](#)」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイドの [IAM でのクロスアカウントリソースアクセス](#) を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられたときにアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「[JSON ポリシー概要](#)」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイドの [カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する](#) を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「[管理ポリシーとインラインポリシーのいずれかを選択する](#)」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、[プリンシパルを指定する](#) 必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「[IAM エンティティのアクセス許可境界](#)」を参照してください。

- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「[サービスコントロールポリシー](#)」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「[リソースコントロールポリシー \(RCP\)](#)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「[セッションポリシー](#)」を参照してください。

複数のポリシータイプ

1つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「[ポリシー評価ロジック](#)」を参照してください。

AWS HealthImaging が IAM で機能する仕組み

IAM を使用して HealthImaging へのアクセスを管理する前に、HealthImaging で利用できる IAM の機能について学びます。

AWS HealthImaging で使用できる IAM の機能

IAM の特徴量	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

IAM の特徴量	HealthImaging サポート
ABAC (ポリシー内のタグ)	部分的
一時認証情報	あり
プリンシパルアクセス権限	あり
サービスロール	あり
サービスリンクロール	不可

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については、IAM ユーザーガイドの[AWS 「IAM と連携する のサービス」](#)を参照してください。

HealthImaging のアイデンティティベースのポリシー

アイデンティティベースのポリシーのサポート: 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ポリシーの作成方法については、「IAM ユーザーガイド」の[「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」](#)を参照してください。

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。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。

HealthImaging のアイデンティティベースのポリシーの例

HealthImaging でのアイデンティティベースのポリシーの例は、「[AWS HealthImaging のアイデンティティベースのポリシーの例](#)」を参照してください。

HealthImaging 内のリソースベースのポリシー

リソースベースのポリシーのサポート: 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コントロールできます。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、[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、全体のアカウント、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。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。

HealthImaging のポリシーアクション

ポリシーアクションのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの Action 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

HealthImaging アクションのリストを確認するには、「サービス認可リファレンス」の「[AWS HealthImaging で定義されるアクション](#)」を参照してください。

HealthImaging のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。

```
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 のアイデンティティベースのポリシーの例は、「[AWS HealthImaging のアイデンティティベースのポリシーの例](#)」を参照してください。

HealthImaging のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、[Amazon リソースネーム \(ARN\)](#) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*" 
```

HealthImaging リソースのタイプとその ARN のリストを確認するには、「サービス認可リファレンス」の「[AWS HealthImaging で定義されるリソースタイプ](#)」を参照してください。どのアクションとリソースで ARN を使用できるかについては、「[AWS HealthImaging で定義されるアクション](#)」を参照してください。

HealthImaging のアイデンティティベースのポリシーの例は、「[AWS HealthImaging のアイデンティティベースのポリシーの例](#)」を参照してください。

HealthImaging の条件キー

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 JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの[条件演算子](#)を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を

一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」の [AWS 「グローバル条件コンテキストキー」](#) を参照してください。

HealthImaging 条件キーのリストを確認するには、「サービス認可リファレンス」の [「AWS HealthImaging の条件キー」](#) を参照してください。どのアクションとリソースで条件キーを使用できるかについては、「[AWS HealthImaging で定義されるアクション](#)」を参照してください。

HealthImaging でのアイデンティティベースのポリシーの例は、「[AWS HealthImaging のアイデンティティベースのポリシーの例](#)」を参照してください。

HealthImaging の ACL

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アクセス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。

HealthImaging での RBAC

RBAC のサポート	はい
------------	----

IAM で使用される従来の認可モデルは、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、AWS の外部でロールとして知られる、ユーザー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。

HealthImaging での ABAC

ABAC (ポリシー内のタグ) のサポート: 一部

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 HealthImagingによるリソースのタグ付け](#) を参照してください。

属性ベースのアクセス制御 (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

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 認可でアクセス許可を定義する](#)」を参照してください。ABAC をセットアップする手順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「[属性ベースの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。

HealthImaging での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報は、AWS リソースへの短期的なアクセスを提供し、フェデレーションまたはスイッチロールの使用時に自動的に作成されます。AWS では、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お勧めします。詳細については、「IAM ユーザーガイド」の「[IAM の一時的な認証情報](#)」および「[AWS のサービスと IAM との連携](#)」を参照してください。

HealthImaging のクロスサービスプリンシパル許可

転送アクセスセッション (FAS) のサポート: あり

IAM ユーザーまたはロールを使用して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アクセス許可が必要です。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、「サービス認可リファレンス」の「[AWS HealthImaging 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。

HealthImaging のサービスロール

サービスロールのサポート: 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 のサービスに許可を委任するロールを作成する](#)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 の機能が阻害される可能性があります。HealthImaging が指示する場合以外は、サービスロールを編集しないでください。

HealthImaging の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。

サービスにリンクされた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「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

AWS HealthImaging のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールには、HealthImaging リソースを作成または変更するアクセス許可はありません。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。

これらのサンプルの JSON ポリシードキュメントを使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「[IAM ポリシーを作成する \(コンソール\)](#)」を参照してください。

各リソースタイプの ARNs「サービス認可リファレンス」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

- [ポリシーに関するベストプラクティス](#)
- [HealthImaging コンソールの使用](#)
- [自分の権限の表示をユーザーに許可する](#)

ポリシーに関する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、または削除できるかどうかを決定します。これらのアクションでは、AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらはで使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイドの [AWS マネージドポリシー](#) または [ジョブ機能のAWS マネージドポリシー](#) を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイドの [IAM でのポリシーとアクセス許可](#) を参照してください。

- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションがなどの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイドの [IAM JSON ポリシー要素:条件](#) を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイドの [IAM Access Analyzer でポリシーを検証する](#) を参照してください。
- 多要素認証 (MFA) を要求する - で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイドの [MFA を使用した安全な API アクセス](#) を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイドの [IAM でのセキュリティのベストプラクティス](#) を参照してください。

HealthImaging コンソールの使用

AWS HealthImaging コンソールにアクセスするには、一連の最小限のアクセス許可が必要です。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引き続き HealthImaging 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 マネージド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:ListGroupForUser",
        "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 AWS HealthImaging の マネージドポリシー

AWS 管理ポリシーは、によって作成および管理されるスタンドアロンポリシーです AWS。AWS 管理ポリシーは、多くの一般的なユースケースにアクセス許可を付与するように設計されているため、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できます。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有の[カスタマー管理ポリシー](#)を定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。AWS は、新しい が起動されるか、新しい API オペレーション AWS のサービス が既存のサービスで使用できるようになったときに、AWS 管理ポリシーを更新する可能性が高くなります。

詳細については、「IAM ユーザーガイド」の「[AWS マネージドポリシー](#)」を参照してください。

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](#)
- [AWS マネージドポリシーに対する HealthImaging の更新](#)

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。

JSON

```
{  
  "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。

JSON

```
{  
  "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": "*"
}
}
}

```

AWS マネージドポリシーに対する HealthImaging の更新

このサービスがこれらの変更の追跡を開始してからの HealthImaging の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知については、「[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 は AWS 、管理ポリシーの変更の追跡を開始しました。	2023 年 7 月 19 日

HealthImaging でのサービス間の混乱した代理の防止

混乱した代理問題は、アクションを実行する許可を持たないエンティティが、より特権のあるエンティティにアクションを実行するように強制できるセキュリティの問題です。AWS では、サービス間でのなりすましが、混乱した代理問題の原因となる可能性があります。サービス間でのなりすまは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、AWS では、アカウントのリソースへのアクセス許可が付与されたサービスプリンシパルを持つすべてのサービスのデータ保護に役立つツールを提供しています。

AWS HealthImaging がリソースに別のサービスに付与するアクセス許可を制限するには、IAM ImportJobDataAccessRole ロールの信頼関係ポリシーで [aws:SourceArn](#) および [aws:SourceAccount](#) グローバル条件コンテキストキーを使用することをお勧めします。1 つのリソースだけをクロスサービスのアクセスに関連付ける場合は、aws:SourceArn を使用します。アカウント内の任意のリソースをクロスサービスの使用に関連付ける場合は、aws:SourceAccount を使用します。これらのグローバル条件コンテキストキーの両方を、同じポリシーステートメントで使用する場合、aws:SourceAccount 値と aws:SourceArn 値の中の参照されるアカウントには、同じアカウント ID を使用する必要があります。

の値は、影響を受けるデータストアの ARN aws:SourceArn である必要があります。データストアの完全な ARN がわからない場合、または複数のデータストアを指定する場合は、ARN の不明な部分に * ワイルドカードで aws:SourceArn グローバルコンテキスト条件キーを使用します。たとえば、aws:SourceArn を arn:aws:medical-imaging:us-west-2:111122223333:datastore/* に設定できます。

次の信頼ポリシーの例では、aws:SourceArn および aws:SourceAccount 条件キーを使用して、データストアの ARN に基づいてサービスプリンシパルへのアクセスを制限し、混乱した代理問題を防ぎます。

AWS HealthImaging の ID とアクセスのトラブルシューティング

次の情報は、HealthImaging と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- [HealthImaging でアクションを実行する権限がない](#)
- [iam:PassRole を実行する権限がありません](#)
- [自分の 以外のユーザーに HealthImaging リソース AWS アカウント へのアクセスを許可したい](#)

HealthImaging でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な *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:PassRole を実行する権限がありません

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 HealthImaging 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。

一部の AWS のサービスでは、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

次の例では、*marymajor* という名前の IAM ユーザーがコンソールを使用して HealthImaging でアクションを実行しようとした際に、エラーが発生しています。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに *iam:PassRole* アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに 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:

- これらの機能が HealthImaging でサポートされるかどうかを確認するには、「[AWS HealthImaging が IAM で機能する仕組み](#)」を参照してください。
- 所有 AWS アカウントしているのリソースへのアクセスを提供する方法については、「[IAM ユーザーガイド](#)」の「[所有 AWS アカウントしている別の IAM ユーザーへのアクセスを提供する](#)」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「[サードパーティー AWS アカウントが所有するへのアクセスを提供する](#)」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイドの [外部で認証されたユーザー \(ID フェデレーション\) へのアクセスの許可](#) を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用法の違いについては、「IAM ユーザーガイド」の「[IAM でのクロスアカウントのリソースへのアクセス](#)」を参照してください。

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コンプライアンスプログラム](#)」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。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「[レポートをダウンロードする AWS Artifact](#)」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](#) – セキュリティとコンプライアンスの自動リファレンスデプロイガイドでは、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デプロイする手順について説明します AWS。

- [「Architecting for HIPAA Security and Compliance」ホワイトペーパー](#) – このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- [での GxP システム AWS](#) – 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 AWS にどのようにアプローチするかに関する情報と、GxP のコンテキストでの AWS サービスの使用に関するガイダンスを提供します。
- [AWS コンプライアンスリソース](#) – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- [ルールによるリソースの評価](#) – リソース設定が内部プラクティス、業界ガイドライン、規制にどの程度準拠しているか AWS Config を評価します。
- [AWS Security Hub CSPM](#) – この AWS サービスは、内のセキュリティ状態を包括的に把握 AWS し、セキュリティ業界標準とベストプラクティスへの準拠を確認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

マネージドサービスである AWS HealthImaging は、ホワイトペーパー [「Amazon Web Services: 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手順で保護されています。

AWS 公開された 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、一時的ディフィー・ヘルマン 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 はと統合されています。これは AWS CloudFormation、AWS リソースとインフラストラクチャの作成と管理に費やす時間を短縮できるように、リソースのモデル化とセットアップに役立つサービスです。必要なすべての AWS リソースを記述するテンプレートを作成し、それらのリソースを CloudFormation プロビジョニングして設定します。

を使用すると CloudFormation、テンプレートを再利用して HealthImaging リソースを一貫して繰り返しセットアップできます。リソースを 1 回記述し、複数の AWS アカウント およびリージョンで同じリソースを何度もプロビジョニングします。

HealthImaging と 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、[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 や 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は、CloudFormation スタックでプロビジョニングするリソースを記述します。JSON または YAML に慣れていない場合は、CloudFormation デザイナーを使用して CloudFormation テンプレートの使用を開始できます。詳細については、「AWS CloudFormation ユーザーガイド」の「[CloudFormation Designer とは](#)」を参照してください。

AWS HealthImaging は、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。

の詳細 CloudFormation

詳細については CloudFormation、次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](#)
- [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)

VPC と の間にプライベート接続を確立するには、インターフェイス VPC エンドポイント AWS HealthImaging を作成します。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、NAT デバイス、VPN 接続、AWS Direct Connect 接続のいずれも必要とせずに DevOps Guru API にプライベートにアクセスできるテクノロジーである [AWS PrivateLink](#) を利用します。VPC のインスタンスは、パブリック IP アドレスがなくても HealthImaging API と通信できます。VPC と HealthImaging 間のトラフィックは、Amazon ネットワークから離れません。

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](#)
- [HealthImaging 用のインターフェイス VPC エンドポイントの作成](#)
- [HealthImaging 用の VPC エンドポイントポリシーの作成](#)

HealthImaging VPC エンドポイントに関する考慮事項

HealthImaging のインターフェイス VPC エンドポイントを設定する前に、「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ています。

HealthImaging 用のインターフェイス VPC エンドポイントの作成

HealthImaging サービス用の VPC エンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface (AWS CLI) を使用して作成できます。詳細については、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。

HealthImaging 用の VPC エンドポイントは、以下のサービス名を使用して作成します。

- com.amazonaws.*region*.medical-imaging
- com.amazonaws.*region*.runtime-medical-imaging
- com.amazonaws.*region*.dicom-medical-imaging

Note

PrivateLink 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。

リージョンのデフォルト DNS 名 (例: medical-imaging.us-east-1.amazonaws.com) を使用して、HealthImaging に API リクエストを作成できます。

詳細については、「Amazon VPC ユーザーガイド」の「[インターフェイスエンドポイントを介したサービスへのアクセス](#)」を参照してください。

HealthImaging 用の VPC エンドポイントポリシーの作成

VPC エンドポイントには、HealthImaging へのアクセスを制御するエンドポイントポリシーをアタッチできます。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 アクション用の VPC エンドポイントポリシー

以下は、HealthImaging 用のエンドポイントポリシーの例です。エンドポイントにアタッチされると、このポリシーは、すべてのリソースですべてのプリンシパルに、HealthImaging アクションへのアクセス権を付与します。

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、他の [AWS Organizations](#) が所有するアカウント、および のオープンデータレジストリにある [Imaging Data Commons \(IDC\)](#) などのオープン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は次のとおりです。

- 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
- 複数機関の臨床試験間でデータを安全に共有する研究者

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) バケット所有者は、HealthImaging データストア所有者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 データストアの所有者は、IAM に Amazon S3 バケットを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、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、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ゾーンを中心に構築されています。は、低レイテンシー、高スループット、高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティゾーン AWS リージョン を提供します。アベイラビリティゾーンでは、ゾーン間で中断することなく自動的にフェールオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、フォールトトレランス、および拡張性が優れています。

データまたはアプリケーションを遠方でレプリケートする必要がある場合は、AWS ローカルリージョンを使用します。AWS ローカルリージョンは、既存の AWS リージョンを補完するように設計された単一のデータセンターです。すべてと同様に AWS リージョン、AWS ローカルリージョンは他のリージョンから完全に分離されています AWS リージョン。

AWS リージョン およびアベイラビリティゾーンの詳細については、[AWS 「グローバルインフラストラクチャ」](#) を参照してください。

AWS HealthImaging 参考資料

Note

すべてのネイティブ HealthImaging API アクションとデータ型については、[AWS HealthImaging API リファレンス](#)を参照してください。

トピック

- [AWS HealthImaging の DICOM サポート](#)
- [AWS HealthImaging リファレンス](#)

AWS HealthImaging の DICOM サポート

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 メタデータキーはこれらに基づいているため、サポートされている患者レベル、治験、シリーズレベルの DICOM データ要素に精通してください。インポートを開始する前に、医療画像データが HealthImaging がサポートする転送構文と DICOM の要素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

- [サポートされている SOP クラス](#)
- [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 HealthImaging を使用すると、[リタイア](#)と[プライベート](#)を含む任意の 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 HealthImaging 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。???変換プロセス中に、HealthImaging メタデータキーは特定のバージョンの DICOM 標準に基づいて生成されます。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 HealthImaging は、患者、治験、シリーズレベルで次の DICOM データ要素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 HealthImaging は、以下の患者レベルの要素をサポートしています。

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 HealthImaging は、以下の治験レベルの要素をサポートしています。

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 HealthImaging は以下のシリーズレベルの要素をサポートしています。

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 HealthImaging では、転送構文が異なる DICOM P10 ファイルのインポートがサポートされています。一部のファイルはインポート中に元の転送構文エンコードを保持し、ほとんどの可逆画像フレームはインポート中にトランスコードされます。トランスコードの動作は、データストアの設定によって異なります。データストアはデフォルトでストレージ形式として HTJ2K を使用しますが、作成時に JPEG 2000 Lossless を使用するように設定できます。次の例は、HealthImaging がによって返される [メタデータ](#) 内のインスタンス `StoredTransferSyntaxUID` ごとに を記録する方法を示しています [GetImageSetMetadata](#)。

```
"Instances": {  
  "999.999.2.19941105.134500.2.101": {  
    "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.90",  
    "ImageFrames": [{ ...
```

Note

次の表を表示するときは、次の点に注意してください。

- アスタリスク (*) が付いた転送構文 UID エントリは、インポート中にファイルが元のエンコード形式で保存されていることを示します。これらのファイルの場合、インスタンスメタデータ `StoredTransferSyntaxUID` にある は元の転送構文と一致します。
- アスタリスクなしでマークされた転送構文 UID エントリは、インポート中にファイルが HTJ2K ロスレスにトランスコードされ、HealthImaging に保存されていることを示します。インスタンスメタデータの `StoredTransferSyntaxUID` 要素は、RPCL オプションイメージ圧縮を使用した高スループット JPEG 2000 のストレージ形式 - 損失のみ (1.2.840.10008.1.2.4.202) に設定されます。
- `StoredTransferSyntaxUID` キーが存在しないか、 に設定されている場合 `null`、設定されたデータストアストレージ形式にエンコードされていると仮定できます。

HealthImaging でサポートされている転送構文

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* (バイナリセグメンテーションは元のエンコードを保持しますが、非バイナリセグメンテーションは HTJ2K Lossless RPCL にトランスコードされます)	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)
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
1.2.840.10008.1.2.4.92	JPEG 2000 Part 2 マルチコンポーネントイメージ圧縮 (ロスレスのみ)

転送構文 UID	転送構文名
1.2.840.10008.1.2.4.93	JPEG 2000 Part 2 マルチコンポーネントイメージ圧縮
1.2.840.10008.1.2.4.112*	JPEG XL
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	高スループット JPEG 2000 と RPCL オプションイメージ圧縮 (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
1.2.840.10008.1.2.4.100*、1.2.840.10008.1.2.4.100.1*	MPEG2 メインプロファイルのメインレベル
1.2.840.10008.1.2.4.101*、1.2.840.10008.1.2.4.101.1*	高レベルの MPEG2 メインプロファイル
1.2.840.10008.1.2.4.102*、1.2.840.10008.1.2.4.102.1*	MPEG-4 AVC/H.264 ハイプロファイル/レベル 4.1
1.2.840.10008.1.2.4.103*、1.2.840.10008.1.2.4.103.1*	MPEG-4 AVC/H.264 BD 互換のハイプロファイル/レベル 4.1
1.2.840.10008.1.2.4.104*、1.2.840.10008.1.2.4.104.1*	2D ビデオ用の MPEG-4 AVC/H.264 ハイプロファイル/レベル 4.2
1.2.840.10008.1.2.4.105*、1.2.840.10008.1.2.4.105.1*	MPEG-4 AVC/H.264 ハイプロファイル/ITU-T H.264 ビデオのレベル 4.2
1.2.840.10008.1.2.4.106*、1.2.840.10008.1.2.4.106.1*	MPEG-4 AVC/H.264 ステレオハイプロファイル/ITU-T H.264 ビデオのレベル 4.2
1.2.840.10008.1.2.4.107*	HEVC/H.265 メインプロファイル/レベル 5.1 ビデオ

転送構文 UID	転送構文名
1.2.840.10008.1.2.4.108*	HEVC/H.265 メイン 10 プロファイル/レベル 5.1

*インポート中に元の転送構文エンコードを保持します

DICOM 要素の制約

医療画像データを AWS HealthImaging にインポートすると、以下の DICOM 要素に最大長制約が適用されます。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

DICOM キーワード	DICOM タグ	最大長
PatientName	(0010,0010)	256
PatientID	(0010,0020)	256
PatientBirthDate	(0010,0030)	18
PatientSex	(0010,0040)	16
StudyInstanceUID	(0020,000D)	256
StudyID	(0020,0010)	256
StudyDescription	(0008,1030)	256
NumberOfStudyRelatedSeries	(0020,1206)	1,000,000
NumberOfStudyRelatedInstances	(0020,1208)	1,000,000
AccessionNumber	(0008,0050)	256
StudyDate	(0008,0020)	18
StudyTime	(0008,0030)	28

DICOM キーワード	DICOM タグ	最大長
SOPInstanceUID	(0008,0018)	256
SeriesInstanceUID	(0020,000E)	256

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、Patient/Study/Series/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 で生成された属性は更新できません:
`SchemaVersion`、`DatastoreID`、`ImageSetID`、`PixelData`、`Checksum`、`Width`、`Height`、`MinPi`
- `force` フラグが設定されていない限り、次の DICOM 属性を更新することはできません:
`Tag.PixelData`、`Tag.StudyInstanceUID`、`Tag.SeriesInstanceUID`、`Tag.SOPInstanceUID`、`Tag.StudyID`
- `force` フラグが設定されていないと、VR タイプ SQ (ネストされた属性) で属性を更新できません
- `force` フラグが設定されていないと、複数值属性を更新できません
- `force` フラグが設定されていない限り、属性 VR タイプと互換性のない値で属性を更新することはできません
- `force` フラグが設定されていない限り、DICOM 標準に従って有効な属性と見なされない属性を更新することはできません
- 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 リファレンス

このセクションでは、AWS HealthImaging に関連するサポートデータについて説明します。

トピック

- [AWS HealthImaging エンドポイントとクォータ](#)
- [AWS HealthImaging のスロットリング制限](#)
- [AWS HealthImaging ピクセルデータの検証](#)
- [HealthImaging 警告コード](#)
- [AWS HealthImaging の画像フレームデコードライブラリ](#)
- [AWS HealthImaging のサンプルプロジェクト](#)
- [AWS SDK でのこのサービスの使用](#)

AWS HealthImaging エンドポイントとクォータ

以下のトピックには、AWS HealthImaging サービスのエンドポイントとクォータに関する情報が含まれています。

トピック

- [サービスエンドポイント](#)
- [サービスクォータ](#)

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、より高速な接続を可能にするために、特定のリージョンのエンドポイントを提供します。次の表は、AWS HealthImaging のサービスエンドポイントを示しています。

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
		medical-imaging-fips.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
			HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
		medical-imaging-fips.us-west-2.amazonaws.com	
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、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せん。

[https://medical-imaging.*region*.amazonaws.com](https://medical-imaging.<i>region</i>.amazonaws.com)

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- StartDICOMImportJob

- GetDICOMImportJob
- ListDICOMImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

次のイメージセットアクションは、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 Retrieve WADO-RS サービスを表します。詳細については、「[HealthImaging から DICOM データを取得する](#)」を参照してください。

エンドポイントから次の DICOMweb サービスにアクセスできます。

```
https://dicom-medical-imaging.region.amazonaws.com
```

- GetDICOMInstance
- GetDICOMInstanceMetadata
- GetDICOMInstanceFrames

サービスクォータ

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。

次の表は、AWS HealthImaging のデフォルトクォータを示しています。

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	サポートされている各リージョン: 100	可 能	現在の AWS リージョンのデータストアあたりの

名前	デフォルト	引き上げ可能	説明
			同時 CopyImageSet リクエストの最大数
データストアあたりの最大同時 DeletelImageSet リクエスト	サポートされている各リージョン: 100	可能	現在の AWS リージョンのデータストアあたりの同時 DeletelImageSet リクエストの最大数
データストアあたりの最大同時 UpdateImageSetMetadata リクエスト	サポートされている各リージョン: 100	可能	現在の AWS リージョンのデータストアあたりの同時 UpdateImageSetMetadata リクエストの最大数
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	あり	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
最大データストア	サポートされている各リージョン: 10	あり	現在の AWS リージョン内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が許可される ImageFrames の最大数	サポートされている各リージョン: 1,000	あり	現在の AWS リージョンで CopyImageSet リクエストごとにコピーできる ImageFrames の最大数

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	あり	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の最大数	サポートされている各リージョン: 10,000	不可	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
UpdateImageSetMetadata によって受け入れられる最大ペイロードサイズ制限 (KB 単位)	サポートされている各リージョン: 10 KB	あり	現在の AWS リージョンで UpdateImageSetMetadata によって受け入れられる最大ペイロードサイズ制限 (KB 単位)
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	不可	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)
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	不可	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)

名前	デフォルト	引き上げ可能	説明
Import、Copy、UpdateImageSet あたりの ImageSetMetadata の最大サイズ制限 (MB 単位)	サポートされている各リージョン: 50 MB	あ り	現在の AWS リージョンにおけるインポート、コピー、および UpdateImageSet あたりの ImageSetMetadata の最大サイズ制限 (MB 単位)

AWS HealthImaging のスロットリング制限

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

すべての HealthImaging API アクションのためにスロットリング制限を調整することが可能です。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。ケースを作成するには、AWS アカウントにログインし、ケースの作成を選択します。

次の表に、[ネイティブ HealthImaging アクション](#)と [DICOMweb サービスの表現の両方](#)のスロットリング制限を示します。

AWS HealthImaging のスロットリング制限

Action	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

Action	スロットリングレート	スロットリングバースト
GetDatastore	10 tps	20 tps
ListDatastores	5 tps	10 tps
DeleteDatastore	0.085 tps	1 tps
StartDICOMImportJob	1 tps	2 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
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
GetDICOMInstance*	50 tps	100 tps
GetDICOMInstanceMetadata*	50 tps	100 tps
GetDICOMInstanceFrames*	50 tps	100 tps

Action	スロットリングレート	スロットリングバースト
GetDICOMSeriesMetadata	50 tps	100 tps

*DICOMwebサービスの表現

AWS HealthImaging ピクセルデータの検証

インポート中、HealthImaging 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HealthImaging 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。

- イメージの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 チェックサム値は `job-output-manifest.json` メタデータドキュメントに表示されます。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。
- イメージが HealthImaging [データストア](#) にインポートされ、[イメージセット](#) に変換されると、HTJ2K-encodedされた [イメージフレーム](#) がすぐにデコードされ、新しい IFRCs が計算されます。HealthImaging は、元のイメージのフル解像度 IFRCs とインポートされたイメージフレームの新しい 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、イメージ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. には、フル解像度イメージの IFRC (チェックサム) `PixelDataChecksumFromBaseToFullResolution` が含まれています。元の転送構

文 1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50、および 1.2.840.10008.1.2.1 (バイナリセグメンテーションのみ) に保存されているイメージの場合、チェックサムは元のイメージで計算されます。RPCL を使用して HTJ2K Lossless に保存されているイメージの場合、チェックサムはデコードされたフル解像度イメージで計算されます。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。

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、に記録される IFRC のメタデータ出力の例です job-output-manifest.json。

```
"ImageFrames": [{
  "ID": "67890678906789012345123451234512",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 512,
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
}]
```

元の転送構文 1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50、および 1.2.840.10008.1.2.1 (バイナリセグメンテーションのみ) に保存されているイメージMaxPixelValueの場合、MinPixelValueおよびは利用できません。は元のフレームのサイズFrameSizeInBytesを示します。

```
"PixelDataChecksumFromBaseToFullResolution": [
  {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": null,
"MaxPixelValue": null,
"FrameSizeInBytes": 429
```

RPCL で HTJ2K Lossless に保存されているイメージの場合、はデコードされたイメージフレームのサイズFrameSizeInBytesを示します。

```
"PixelDataChecksumFromBaseToFullResolution": [
  {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": 11,
"MaxPixelValue": 11,
"FrameSizeInBytes": 1652
```

4. ビデオを含むインスタンスの場合、HealthImaging は、DICOM メタデータで指定された転送構文がビデオコーデックと一致することを確認する軽量コーデック検証を実行します。

HealthImaging は、CRC32 アルゴリズムを使用して元のビデオオブジェクトの IFRC チェックサム値を計算します。IFRC チェックサム値は `job-output-manifest.json`、HealthImaging メタデータに保持されます。元の転送構文 (上記) に保存されているイメージと同様に `MaxPixelValue`、`MinPixelValue` とは使用できません。`FrameSizeInBytes` は、元のフレームのサイズを示します。

5. HealthImaging はピクセルデータを検証し、GitHub の [ピクセルデータ検証](#) 手順にアクセスし、ファイルの指示 `README.md` に従って、HealthImaging が使用するさまざまな [可逆画像処理](#) を個別に検証 [イメージフレームデコードライブラリ](#) します。完全なイメージがロードされたら、最後に raw 入力データの IFRC を計算し、HealthImaging メタデータで指定された IFRC 値と比較してピクセルデータを検証できます。

HealthImaging 警告コード

HealthImaging は、すべての医療画像データのインポートを試みます。インポート中にデータの非準拠または認識されないデータ要素が発生した場合、HealthImaging は `warning.ndjson` ファイルに次のいずれかの警告を追加します。インポートされたインスタンスに関連付けられた警告は、`WarningReason` 要素の `SearchDICOMInstances` アクションを介して検索することもできます。警告付きでインポートされたインスタンスでは、以下に示すように、HealthImaging APIs によるサポートが減る可能性があります。

HealthImaging インポート警告コード

警告の理由 (16 進数)	警告の理由 (10 進数)	警告タイプ (列挙型)	警告の詳細	結果として生じる動作
------------------	------------------	----------------	-------	------------

DICOM 標準警告の理由

0xB000	45056	COERCION_OF_DATA_ELEMENTS	取り込みにより、インスタンスのストレージ中に 1 つ以上のデータ要素が変更されました。セクション 6.6.1.3 を参照してください。	該当なし
--------	-------	---------------------------	---	------

警告の理由 (16 進数)	警告の理由 (10 進数)	警告タイプ (列挙型)	警告の詳細	結果として生じる動作
0xB006	45062	ELEMENTS_ DISCARDED	取り込みにより、インスタンスのストレージ中に一部のデータ要素が破棄されました。セクション 6.6.1.3 を参照してください。	該当なし
0xB007	45063	SOP_CLASS_ _DATA_MIS MATCH	このStoreDICOM アクションでは、インスタンスのストレージ中にデータセットが SOP クラスの制約と一致しないことが確認されました。	該当なし

AWS HealthImaging の警告理由

0xB100	45312	TRANSCODI NG_EXCEPT ION	この警告は、HealthImaging がインスタンス PixelData を HTJ2K にトランスコードできない場合 (デフォルトのストレージ形式)、または別の理由 (検証の失敗、誤ったピクセルデータ属性など) で PixelData をトランスコードできない場合に発生します。この場合、ピクセルデータは BLOB として保存されます。	ピクセルデータは、必要に応じて 1 つの BLOB として取得できます。Accept ヘッダー transfer-syntax のとしてワイルドカード「*」を渡すと、保存された形式で返されます。
--------	-------	-------------------------------	--	---

警告の理由 (16 進数)	警告の理由 (10 進数)	警告タイプ (列挙型)	警告の詳細	結果として生じる動作
0xB110	45328	FRAMES_EXTRACTION_FAILURE	この警告は、指定された DICOM メタデータに基づいて PixelData から個々のフレームを解析する際に問題が発生した場合に発生します。	ピクセルデータが形式が正しくないため取得できません。GetDICOMInstance を使用してインスタンス全体を取得します。
0xB111	45329	FRAME_NUMBER_MISMATCH	この警告は、「NumberOfFrames」DICOM 要素が、入力 DICOM ファイル内の画像の「フラグメント」の実際の数と一致しない場合に発生します。	ピクセルデータが形式が正しくないため取得できません。GetDICOMInstance を使用してインスタンス全体を取得します。
0xB112	45330	INVALID_OFFSET_TABLE	この警告は、入力 DICOM ファイルのフラグメントのオフセットテーブルが実際のフレーム長と一致せず、重大度に応じてフレームの形式が正しくない可能性がある場合に発生します。	ピクセルデータが形式が正しくないため取得できません。GetDICOMInstance を使用してインスタンス全体を取得します。
0xB120	45344	UNSUPPORTED_SYNTAX	この警告は、HealthImaging が認識されない、またはサポートされていない転送構文を検出した場合に発生します。この場合、HealthImaging はピクセルデータを BLOB として保存します。	ピクセルデータは、必要に応じて 1 つの BLOB として取得できます。AcceptHeader-transfer-syntax のとしてワイルドカード「*」を渡すと、保存された形式で返されます。

警告の理由 (16 進数)	警告の理由 (10 進数)	警告タイプ (列挙型)	警告の詳細	結果として生じる動作
0xB201	45570	INVALID_UID_FORMAT	この警告は、1 つ以上の UID 要素が DICOM 値表現 (例: 1.2.3..4) に違反した場合に発生します。	該当なし
0xB202	45571	INVALID_DICOM_VALUE_LENGTH	この警告は、DICOM 要素の長さが DICOM 値表現でサポートされている長さよりも長く、検索/取得アクションに無効な動作が発生する可能性がある場合に発生します。	一部のフィールドは解析できないため、検索できない場合があります (例: StudyDate または StudyTime)
0xBFFF	47513	OTHER	この警告は、HealthImaging が特定の警告コードとしてキャプチャしないという未知の警告がある場合に発生します。	ピクセルデータが形式が正しくないため取得できません。GetDICOMInstance を使用してインスタンス全体を取得します。

AWS HealthImaging の画像フレームデコードライブラリ

[インポート](#) 中、一部の転送構文は元のエンコードを保持し、他の転送構文はデータストアの設定に応じて、デフォルトで高スループット JPEG 2000 (HTJ2K) ロスレスまたは JPEG 2000 ロスレス (設定されている場合) にトランスコードされます。HTJ2K は、HTJ2K の高度な機能に一貫して高速な画像表示と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 または JPEG 2000 Lossless 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。転送構文の決定については、「サポートされている転送構文」を参照してください。転送構文の決定については、「」を参照してください [サポートされる転送構文](#)。

Note

HTJ2K は、[JPEG2000 標準 \(ISO/IEC 15444-15:2019\) 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、プリシント、タイリング、高ビット深度、複数チャンネル、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

- [イメージフレームデコードライブラリ](#)
- [イメージビューワー](#)

イメージフレームデコードライブラリ

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
- [カカドウ・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 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。

[OIDC を介して AWS HealthImaging に統合された OHIF Viewer](#)

この[AWS Cloud Development Kit \(AWS CDK\)](#)プロジェクトは、[Amazon CloudFront](#) に [OHIF ビューワー](#)をデプロイします。ビューワーは DICOMWeb データソースとして Amazon Web Services データストアに統合され、OIDC を介した認証の ID プロバイダーとして [Amazon Cognito](#) と統合されます。

[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、によって管理されるエッジで実行されているコンポーネントと[AWS IoT Greengrass](#)、AWS クラウドで実行されている DICOM 取り込みパイプラインで構成されます。

[タイルレベルマーカ \(TLM\) プロキシ](#)

High-Throughput JPEG 2000 (HTJ2K) の機能であるタイルレベルマーカ (TLM) を使用して AWS HealthImaging から画像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](#)

(GET を使用して) キャッシュし、エッジからイメージフレームを配信する HTTPS エンドポイントを使用して [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 ビューア 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画像フレーム (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 (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。

[AWS HealthImaging DICOMweb Proxy](#)

HealthImaging データストアで DICOMweb WADO-RS エンドポイントと "DO-RS エンドポイント" を有効にして、ウェブベースの医療画像ビューワーやその他の DICOMweb 互換アプリケーションをサポートするための Python ベースのプロジェクト。

Note

このプロジェクトは、「」で説明されている DICOMweb APIs の HealthImaging 表現を使用しません [AWS HealthImaging での DICOMweb の使用](#)。

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 でのこのサービスの使用

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

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	AWS Tools for PowerShell コード例

SDK ドキュメント	コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

可用性の例

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

コスト最適化

HealthImaging は、アクセスパターンが変化したときに最も費用対効果の高いアクセス階層にデータを自動的に移動することで、ストレージコストを最適化するように設計されています。使用パターンが時間の経過とともに変化するにつれて、インテリジェント階層化は運用上のオーバーヘッドなしで DICOM データの自動ライフサイクル管理を提供します。HealthImaging には 2 つのストレージ階層があります。

- 新しくインポートされた、または定期的にアクセスされる DICOM データの高頻度アクセス階層ストレージ。
- 最近アクセスされていない DICOM データのインスタントアクセス階層ストレージをアーカイブします。ミリ秒単位のアクセスレイテンシーを維持しながら、長期アーカイブのストレージコストを削減します。

データストア内のすべてのイメージセットの合計ストレージサイズに対して課金されます。どちらのストレージ階層も 1 か月あたり GB ごとに課金され、イメージセットごとに料金は発生しません。また、ストレージ階層間でデータを移動しても取得料金はかかりません。

Note

画像セットは、5 MB 以上のサイズで請求されます。HealthImaging は 5 MB 未満のイメージセットを受け入れますが、これらの小さなオブジェクトは 5 MB のレートで課金されます。

インテリジェント階層化の仕組み

新しいイメージセットが作成されると、データは高頻度アクセス階層に保存されます。画像セットを作成するには、新しいデータをインポートするか (インポートジョブまたは DICOMweb STOW-RS を使用)、CopyImageSet を呼び出します。HealthImaging は、30 日間連続してアクセスされていない画像セットをアーカイブインスタントアクセス階層に自動的に移動し、画像セットは再びアクセスされるまでアーカイブインスタントアクセス階層に残ります。

以下は、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画像セットを自動的に移動する DICOM データへのアクセスを構成するアクションです。

- [GetImageSetMetadata](#)、[GetImageFrame](#)、または [DICOMweb WADO-RS](#) アクションの呼び出し。

- [CopyImageSet](#) または [UpdateImageSetMetadata](#) の呼び出し。コピーオペレーションの場合、レプリケートされたイメージセットのみが高頻度アクセスに階層化されます。Copy with destination の場合、送信先イメージセットは階層化されます。
- AWS HealthImaging マネジメントコンソールを使用してイメージセットデータを表示およびダウンロードする。

上記のアクションは、イメージセットを高頻度アクセス階層に昇格させ、高頻度アクセス階層のイメージセットがアーカイブインスタントアクセス階層にさらに 30 日間階層化されないようにします。イメージセットへのアクセスは、AWS マネジメントコンソールまたは AWS CLI や AWS SDKs などのプログラムインターフェイスを介して行うことができます。他のアクションはアクセスを構成しないため、オブジェク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自動的に移動しません。次に示すのは、そのようなアクションのリストのサンプルであり、決定的なものではありません。

- データストアでのアクション ([CreateDatastore](#) と [GetDatastore](#))
- リストアクション ([ListDICOMImportJobs](#)、[ListImageSetVersions](#)、[ListTagsForResource](#))
- 検索アクション ([SearchImageSets](#) および [DICOMweb の "DO-RS" クエリ](#))
- [GetImageSet](#)、[TagResource](#)、または [UntagResource](#) の呼び出し
- 削除オペレーション

HealthImaging にインポートされたデータは、最小ストレージ期間 30 日間課金されます。データはいつでも削除できますが、インポート後 30 日より前に削除された各イメージには、最小期間の残り分が課金されます。

構造化データストレージの見積もり

HealthImaging は、一部の DICOM ヘッダー情報をインデックス付きストレージに保存します。イメージセットストレージ階層とは無関係に、この構造化ストレージの消費量に対して課金されます。これらの料金は加算されます。データストア内の DICOM リソースの数にリソースあたりのインデックス付きレコードサイズを掛けることで、構造化ストレージの消費量を見積もることができます。次の値はおおよその値です。

DICOM リソース	インデックス付きサイズ (バイト)
研究	1024

DICOM リソース	インデックス付きサイズ (バイト)
列	830
インスタンス	680

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と更新がいつリリースされたかを示しています。詳細については、「リンク先のトピック」を参照してください。

変更	説明	日付
AWS HealthImaging データストアの JPEG XL サポート	HealthImaging は、JPEG XL 転送構文 (1.2.840.10008.1.2.4.112) での DICOM 損失ファイルのインポートと保存をサポートしています。詳細については、 「サポートされている転送構文」 を参照してください。	2026 年 2 月 2 日
非標準 DICOM データの警告サポートによる DICOM インポートの強化	HealthImaging は、EventBridge イベントを通じて詳細な警告を提供しながら、不適合ファイルを受け入れることで、以前に拒否された DICOM データをインポートするようになりました。インポートジョブで、不適合データの特定の警告理由コードを持つ warning.ndjson ファイルを含む WARNING フォルダが生成されるようになりました。この更新では、検索可能な WarningReason (0008,1196) DICOM 要素のサポートが追加され、トランスコードが不可能な場合に保存された形式でインスタンスを取得するための transfer-syntax=* パラメータが導	2025 年 12 月 8 日

入されました。詳細については、[「インポートジョブについて」](#)と[HealthImaging 警告コード](#)」を参照してください。

[AWS HealthImaging データストアの JPEG 2000 ロスレスサポート](#)

AWS HealthImaging は、医療画像のネイティブ JPEG 2000 ロスレスエンコーディングをサポートするようになりました。これにより、トランスコードなしで JPEG 2000 形式でロスレスイメージフレームを永続化および取得するデータストアを作成できます。これにより、[データストアの作成](#)—lossless-storage-format JPEG_2000_LOSSLESS 時に を指定するときに JPEG 2000 Lossless 形式を必要とするアプリケーションのレイテンシーを短縮できます。

2025 年 11 月 25 日

["DO-RS Search Enhancements" 設計](#)

HealthImaging は、未完了またはスペルミスの語句に対応するため、主要な DICOM 属性 (患者名、紹介医師名、患者 ID、治験の説明、モダリティ、およびアクセッション番号) に対するワイルドカード検索と、患者/紹介医師名に対するあいまい検索機能をサポートするようになりました。また、この更新により、患者の生年月日と治験の説明の検索を含めるように "DO-RS API クエリ機能も拡張され、関連する治験とシリーズを見つける機能が強化されました。詳細については、[HealthImaging での DICOM データの検索](#)」を参照してください。

2025 年 9 月 15 日

[DICOMweb APIs の OpenID Connect \(OIDC\) 認可](#)

HealthImaging は、すべての DICOMweb APIs で OpenID Connect (OIDC) ベアラートークン認可をサポートするようになりました。これにより、Authorization ヘッダーで IdP が発行した JWTs を受け入れることで、一般的なビューワーやツールキット (OHIF、SLIM、MONAI など) との標準ベースの OAuth 2.0 の相互運用性が追加されます。詳細については、[DICOMweb APIs](#)」を参照してください。

2025 年 9 月 3 日

[DICOMweb データインポート と DICOMweb Bulkdata のサ ポート](#)

HealthImaging HealthImaging は、DICOMweb STOW-RS プロトコルを介した DICOMweb P10 ファイルの保存をサポートしています。詳細については、[「データのインポート」](#)を参照してください。さらに、HealthImaging は DICOM Bulkdata をサポートしているため、一貫した低レイテンシーのメタデータの取得が保証されます。詳細については、[「DICOM バルクデータの取得」](#)を参照してください。

2025 年 6 月 30 日

[自動データ整理、DICOMweb OktaDO-RS 検索、DICOMweb WADO-RS の機能強化](#)

HealthImaging は、DICOM 標準の患者、治験、シリーズレベルの階層に従って、インポート時に DICOM P10 データを自動的に整理します。詳細については、[「インポートジョブについて」](#)を参照してください。HealthImaging は、DICOMweb の "DO-RS 標準に従ってリッチ検索をサポートしています。詳細については、[HealthImaging での DICOM データの検索](#)」を参照してください。HealthImaging では、「Getting DICOM series metadata [from HealthImaging](#)」で説明されているように、単一の API アクションを介してシリーズ内のすべての DICOM インスタンスのメタデータを取得できません。

2025 年 5 月 22 日

[画像セットの作成で使用される DICOM 要素の数を減らす](#)

HealthImaging は、受信 DICOM P10 オブジェクトを画像セットにグループ化するとき使用される要素の数を減らします。詳細については、[「イメージセットとは」](#)を参照してください。

2025 年 1 月 27 日

[StartDICOMImportJob の可逆サポート](#)

HealthImaging は、DICOM 損失ファイル (1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50) とバイナリセグメンテーションファイルのインポートと保存を元の形式でサポートしています。詳細については、[「サポートされている転送構文」](#)を参照してください。

2024 年 11 月 1 日

[DICOMweb 取り出し APIs の損失サポート](#)

HealthImaging は、1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50、および 1.2.840.10008.1.2.1 (バイナリセグメンテーションのみ) に保存されているイメージとインスタンスを、元の形式または明示的な VR リトルエンディアン (1.2.840.1008.1.2.1) で取得することをサポートしています。詳細については、[「サポートされている転送構文」](#)および[「DICOM データの取得」](#)を参照してください。

2024 年 11 月 1 日

[デジタルスペクティブのインポートの高速化](#)

HealthImaging は、DICOM デジタルスペクトロジー (WSI) のインポートジョブを最大 6 倍高速化します。

2024 年 11 月 1 日

[バイナリセグメンテーションのサポート](#)

HealthImaging は、DICOM バイナリセグメンテーションファイルの取り込みと取得の両方をサポートしています。詳細については、「[サポートされている転送構文](#)」を参照してください。

2024 年 11 月 1 日

[以前のイメージセットバージョン ID に戻す](#)

HealthImaging は、以前のイメージセットバージョン ID に戻すための `revertToVersionId` パラメータを提供します。詳細については、「AWS HealthImaging API リファレンス」の「[revertToVersionId](#)」を参照してください。

2024 年 7 月 24 日

イメージセット変更の強制機能

2024 年 7 月 24 日

HealthImaging は、オプションの forced リクエストパラメータを使用して overrides データ型を提供します。このパラメータを設定する UpdateImageSetMetadata と、患者、治験、またはシリーズレベルのメタデータが一致しなくても、アクションと CopyImageSet アクションが強制されます。詳細については、AWS HealthImaging API リファレンスの「[オーバーライド](#)」を参照してください。

- UpdateImageSetMetadata 強制機能 — HealthImaging では、次の属性を更新するためのオプションの force リクエストパラメータが導入されています。
- Tag.StudyInstanceUID、Tag.SeriesInstanceUID、Tag.SOPInstanceUID、および Tag.StudyID
- インスタンスレベルのプライベート DICOM データ要素の追加、削除、または更新

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。

- CopyImageSet 強制機能 — HealthImaging では、イメージセットをコピーするためのオプションのforceリクエストパラメータが導入されています。このパラメータを設定すると、患者、治験、またはシリーズレベルのメタデータが sourceImageSet と間で一致しない場合でも、CopyImageSet アクションが強制されます destinationImageSet 。このような場合、整合性のないメタデータは で変更されません destinationImageSet 。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。

[SOP インスタンスのサブセットをコピーする](#)

HealthImaging は CopyImageSet アクションを強化し、から 1 つ以上の SOP インスタンスを選択して sourceImageSet にコピーできるようにします destinationImageSet 。詳細については、「[イメージセットのコピー](#)」を参照してください。

2024 年 7 月 24 日

[GetDICOMInstanceMetadata DICOM インスタンスメタデータを返す](#)

HealthImaging は、DICOM Part 10 メタデータ (.json ファイル) を返す GetDICOMInstanceMetadata API を提供します。詳細については、「[インスタンスメタデータの取得](#)」を参照してください。

2024 年 7 月 11 日

[GetDICOMInstanceFrames DICOM インスタンスフレームを返す \(ピクセルデータ\)](#)

HealthImaging は、DICOM Part 10 フレーム (multipart リクエスト) を返す GetDICOMInstanceFrames API を提供します。詳細については、「[インスタンスフレームの取得](#)」を参照してください。

2024 年 7 月 11 日

非標準 DICOM データインポートのサポートを強化

2024 年 6 月 28 日

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、 Treatment Session UIDManufacturer's Device Class UID、 および では Device UID、 次の構文バリエーションを使用できません Acquisition UID 。
- 任意の UID の最初の要素はゼロにすることができません
- UIDs は 1 つ以上の先頭にゼロを付けることができます
- UIDs の長さは最大 256 文字です

[イベント通知](#)

HealthImaging は Amazon EventBridge と統合され、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 の使用](#)」を参照してください。

2024 年 6 月 5 日

[GetDICOMInstance DICOM インスタンスデータを返す](#)

HealthImaging は、DICOM Part 10 インスタンスデータ (.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、「[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ています。詳細については、「[クロスアカウントインポート](#)」を参照してください。

2024 年 5 月 15 日

[イメージセットの検索機能強化](#)

HealthImaging SearchImageSets アクションは、次の検索機能強化をサポートしています。詳細については、[「イメージセットの検索」](#)を参照してください。

2024 年 4 月 3 日

- 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 Descending でソートする
- DICOM シリーズパラメータはレスポンスで返されます

[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 Lossless 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、[「サポートされている転送構文」](#)を参照してください。

2024 年 2 月 16 日

- 1.2.840.10008.1.2.4.57 — JPEG Lossless Non-Hierarchical (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 オプションイメージ圧縮を使用した高スループット JPEG 2000 (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

[テスト済みのコード例](#)

HealthImaging ドキュメントには、Python、JavaScript、Java、C++ 用の AWS CLI および 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、[「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 回のインポートジョブで最大 5,000 件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 回のインポートジョブで最大 10,000 件の入れ子になったフォルダーをサポートします。詳細については、 「サービスクォータ」 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 を使用することで、サポートされているすべてのリージョンでインポートが 20 倍速くなります。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 のサポート	HealthImaging Infrastructure as Code (IaC) でデータストアのプロビジョニングができるようになりました。詳細については、 「を使用した HealthImaging リソースの作成 CloudFormation」 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)、および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 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。