aws

Entwicklerhandbuch

AWS HealthImaging

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

AWS HealthImaging: Entwicklerhandbuch

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

Die Handelsmarken und die Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist AWS HealthImaging? 1
Wichtiger Hinweis 2
Features 2
Zugehörige Services 4
Zugriff 5
HIPAA 5
Preisgestaltung 6
Erste Schritte 7
Konzepte 7
Datastore 7
Bilddatensatz 8
Metadaten 8
Bildrahmen 8
Einrichtung 9
Melde dich an für eine AWS-Konto 9
Erstellen Sie S3-Buckets 9
Einen Datenspeicher erstellen 10
Erstellen eines IAM-Benutzers 10
Erstellen einer IAM-Rolle 11
Installiere das AWS CLI 14
Tutorial 15
Verwaltung von Datenspeichern 16
Einen Datenspeicher erstellen 16
Eigenschaften des Datenspeichers abrufen 25
Datenspeicher auflisten 33
Löschen eines Datenspeichers 40
Verwenden DICOMweb 48
Speichern von Instanzen mit STOW-RS 48
Daten mit WADO-RS abrufen 52
Rufen Sie eine Instanz ab 53
Abrufen von Instance-Metadaten 56
Metadaten der Serie abrufen 57
Frames abrufen 59
Massendaten abrufen 61

Daten werden gesucht mit QIDO-RS ... 63
DICOMWeb-Such-APIs für HealthImaging ... 64
Unterstützte DICOMWeb-Abfragetypen für HealthImaging ... 65
Verwendung IncludeField in QIDO-RS Abfragen ... 69
Suchen Sie nach Studien ... 81
Suchen Sie nach Serien ... 82
Suchen Sie nach Instanzen ... 84
OIDC-Authentifizierung ... 85
So funktioniert die Token-Verifizierung ... 85
Anforderungen und Einrichtung ... 90
Imaging-Daten importieren ... 103
Importaufträge verstehen ... 103
Einen Importjob starten ... 107
Eigenschaften von Importaufträgen abrufen ... 116
Importaufträge auflisten ... 123
Zugreifen auf Bilddatensätze ... 130
Bilddatensätze verstehen ... 130
Suche nach Bilddatensätzen ... 137
Eigenschaften von Bilddatensätzen abrufen ... 163
Metadaten von Bilddatensätzen abrufen ... 170
Pixeldaten des Bildsatzes abrufen ... 181
Ändern von Bilddatensätzen ... 190
Versionen von Bildsätzen auflisten ... 190
Metadaten von Bilddatensätzen aktualisieren ... 197
Um die Metadaten eines primären Bilddatensatzes zu aktualisieren ... 217
Um einen Bildsatz, der nicht primär ist, primär zu machen ... 219
Kopieren eines Bilddatensatzes ... 219
Löschen eines Bilddatensatzes ... 235
Taggen von Ressourcen ... 243
Eine Ressource taggen ... 243
Tags für eine Ressource auflisten ... 249
Eine Ressource entkennzeichnen ... 254
Codebeispiele ... 261
Grundlagen ... 262
Hallo HealthImaging ... 263
Aktionen ... 268

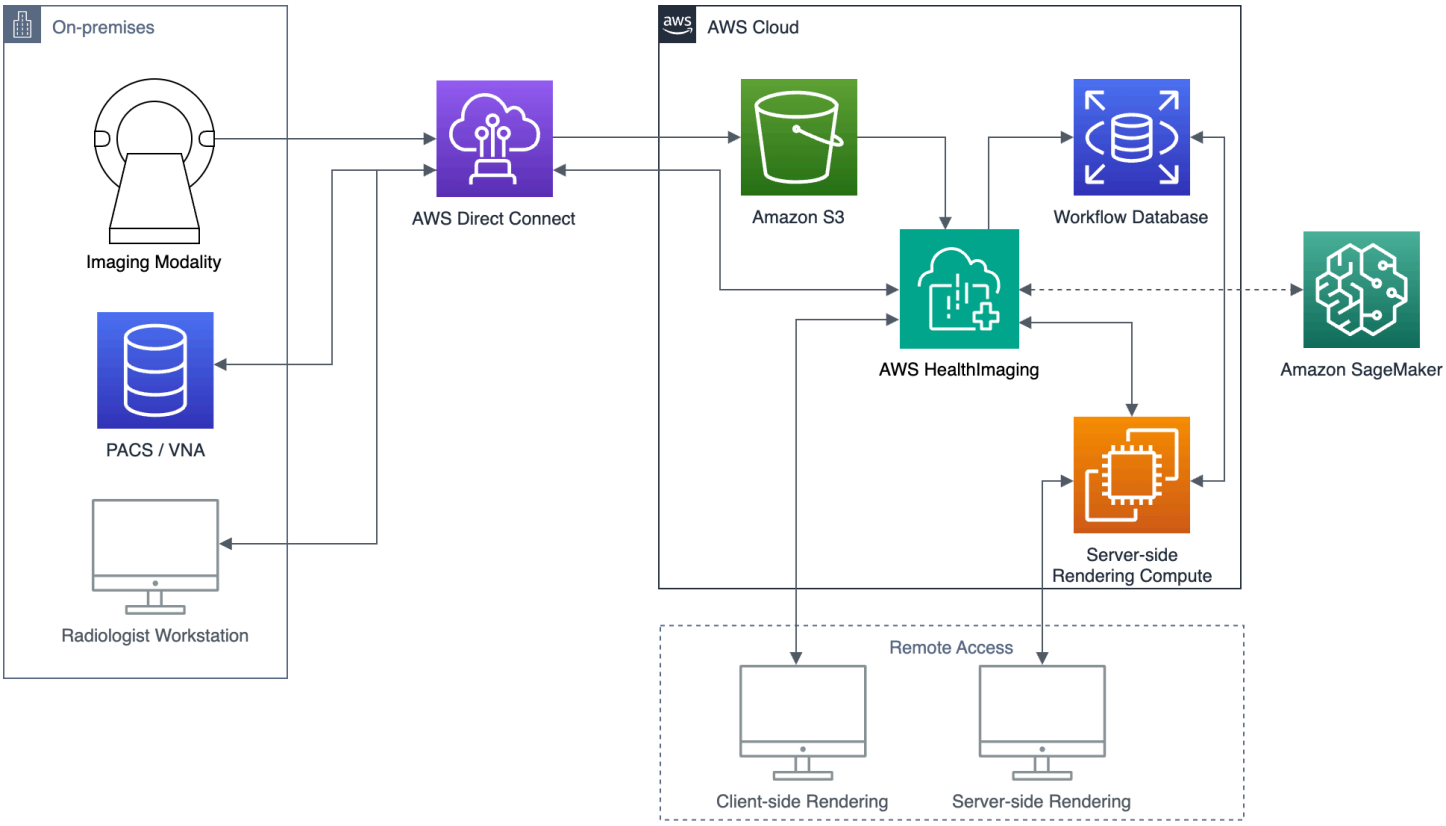
Szenarien ... 422
Erste Schritte mit Bildsätzen und Bild-Frames ... 422
Taggen eines Datenspeichers ... 477
Taggen eines Imagesatzes ... 487
Überwachen ... 498
CloudTrail (API-Aufrufe) ... 499
Einen Trail erstellen ... 499
Grundlegendes zu Protokolleinträgen ... 501
CloudWatch (Metriken) ... 502
API-Metriken ... 503
HealthImaging Metriken ... 503
Dimensionen ... 507
Zugreifen auf -Metriken ... 508
Metriken einrichten ... 508
HealthImaging Metriken anzeigen ... 508
Erstellen eines -Alarms ... 509
EventBridge (Ereignisse) ... 509
HealthImaging Ereignisse wurden gesendet an EventBridge ... 509
HealthImaging Struktur und Beispiele für Ereignisse ... 511
Sicherheit ... 528
Datenschutz ... 529
Datenverschlüsselung ... 530
Datenschutz im Netzwerkverkehr ... 541
Identitäts- und Zugriffsverwaltung ... 541
Zielgruppe ... 542
Authentifizierung mit Identitäten ... 542
Verwalten des Zugriffs mit Richtlinien ... 544
So HealthImaging arbeitet AWS mit IAM ... 546
Beispiele für identitätsbasierte Richtlinien ... 552
AWS verwaltete Richtlinien ... 560
Serviceübergreifende Confused-Deputy-Prävention ... 563
Verwenden von servicegebundenen Rollen ... 565
Fehlerbehebung ... 566
Compliance-Validierung ... 568
Sicherheit der Infrastruktur ... 569
Infrastructure as Code ... 569

HealthImaging und CloudFormation Vorlagen ... 570
Erfahren Sie mehr über CloudFormation ... 570
VPC-Endpunkte ... 570
Überlegungen zu VPC-Endpunkten ... 571
Erstellung eines VPC-Endpunkts ... 571
Erstellen einer VPC-Endpunktrichtlinie ... 572
Kontoübergreifender Import ... 573
Ausfallsicherheit ... 575
Referenz ... 576
DICOM ... 576
Unterstützte SOP-Klassen ... 577
Normalisierung von Metadaten ... 577
Unterstützte Übertragungssyntaxen ... 582
Einschränkungen für DICOM-Elemente ... 585
Einschränkungen für DICOM-Metadaten ... 586
HealthImaging ... 587
Endpunkte und Kontingente ... 587
Drosselungsgrenzen ... 593
Überprüfung der Pixeldaten ... 595
Warncodes ... 597
Bibliotheken zur Dekodierung von Bildrahmen ... 601
Beispielprojekte ... 603
Arbeitet mit AWS SDKs ... 604
Kostenoptimierung ... 606
So funktioniert Intelligentes Tiering ... 606
Schätzung des strukturierten Datenspeichers ... 607
Versionen ... 609
... dcxxix

Was ist AWS HealthImaging?

AWS HealthImaging ist ein HIPAA-fähiger Service, der es Gesundheitsdienstleistern, Life-Science-Organisationen und ihren Softwarepartnern ermöglicht, medizinische Bilder im Petabyte-Bereich in der Cloud zu speichern, zu analysieren und zu teilen. HealthImaging Zu den Anwendungsfällen gehören:

- Bildgebung für Unternehmen — Speichern und streamen Sie Ihre medizinischen Bildgebungsdaten direkt aus der AWS Cloud und behalten Sie gleichzeitig die Leistung mit niedriger Latenz und die hohe Verfügbarkeit bei.
- Langfristige Bildarchivierung — Sparen Sie Kosten bei der langfristigen Bildarchivierung und behalten Sie gleichzeitig den Zugriff auf Bilder in Sekundenschnelle.
- KI/ML-Entwicklung — Lassen Sie mithilfe anderer Tools und Services Inferenzen mit künstlicher Intelligenz und maschinellem Lernen (KI/ML) über Ihr Bildarchiv laufen.
- Multimodale Analyse — Kombinieren Sie Ihre klinischen Bilddaten mit AWS HealthLake (Gesundheitsdaten) und AWS HealthOmics (Omics-Daten), um Erkenntnisse für die Präzisionsmedizin zu gewinnen.



AWS HealthImaging bietet Zugriff auf Bilddaten (z. B. X-Ray, CT, MRT, Ultraschall), sodass in der Cloud erstellte medizinische Bildgebungsanwendungen eine Leistung erzielen können, die bisher nur vor Ort möglich war. Mit reduzieren Sie die Infrastrukturkosten HealthImaging, indem Sie Ihre medizinischen Bildgebungsanwendungen in großem Umfang ausführen, und zwar von einer einzigen, verlässlichen Kopie jedes medizinischen Bildes aus. AWS Cloud

Themen

- Wichtiger Hinweis
- Funktionen von AWS HealthImaging
- Verwandte Dienste AWS
- Zugreifen auf AWS HealthImaging
- HIPAA-Eignung und Datensicherheit
- Preisgestaltung

Wichtiger Hinweis

AWS HealthImaging ist kein Ersatz für professionelle medizinische Beratung, Diagnose oder Behandlung und ist nicht dazu bestimmt, Krankheiten oder Gesundheitsprobleme zu heilen, zu behandeln, zu mildern, zu verhindern oder zu diagnostizieren. Sie sind dafür verantwortlich, im Rahmen der Nutzung von AWS Untersuchungen am Menschen durchzuführen HealthImaging, auch in Verbindung mit Produkten von Drittanbietern, die als Grundlage für klinische Entscheidungen dienen sollen. AWS HealthImaging sollte nur in der Patientenversorgung oder in klinischen Szenarien eingesetzt werden, nachdem es von geschultem medizinischem Fachpersonal mit fundiertem medizinischem Urteilsvermögen überprüft wurde.

Funktionen von AWS HealthImaging

AWS HealthImaging bietet die folgenden Funktionen.

Entwicklerfreundliche DICOM-Metadaten

AWS HealthImaging vereinfacht die Anwendungsentwicklung, indem es DICOM-Metadaten in einem entwicklerfreundlichen Format zurückgibt. Nach dem Import Ihrer Bilddaten ist der Zugriff auf einzelne Metadatenattribute mit benutzerfreundlichen Schlüsselwörtern statt mit unbekannten Hexadezimalzahlen möglich. group/element DICOM-Elemente auf Patienten-,

Studien- und Serienebene sind normalisiert, sodass sich Anwendungsentwickler nicht mehr mit Inkonsistenzen zwischen SOP-Instances auseinandersetzen müssen. Darüber hinaus sind Metadaten-Attributwerte in systemeigenen Laufzeittypen direkt zugänglich.

SIMD-beschleunigte Bilddekodierung

AWS HealthImaging gibt Bildframes (Pixeldaten) zurück, die als High Throughput JPEG 2000 (HTJ2K) -Bilder codiert sind, ein erweiterter Bildkomprimierungscodec. HTJ2K nutzt die Vorteile von Single Instruction Multiple Data (SIMD) auf modernen Prozessoren, um ein neues Leistungsniveau zu erreichen. HTJ2K ist um eine Größenordnung schneller als JPEG2 000 und mindestens doppelt so schnell wie alle anderen DICOM-Übertragungssyntaxen. WASM-SIMD kann verwendet werden, um diese extreme Geschwindigkeit auf Webviewer ohne Platzbedarf zu bringen. Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen.

Überprüfung der Pixeldaten

AWS HealthImaging bietet eine integrierte Überprüfung von Pixeldaten, indem der Status der verlustfreien Kodierung und Dekodierung jedes Bilds während des Imports überprüft wird. Weitere Informationen finden Sie unter Überprüfung der Pixeldaten.

Branchenführende Leistung

AWS HealthImaging setzt dank seiner effizienten Metadatenkodierung, der verlustfreien Komprimierung und des Datenzugriffs mit progressiver Auflösung einen neuen Standard für die Leistung beim Laden von Bildern. Durch die effiziente Metadatenkodierung können Bildbetrachter und KI-Algorithmen den Inhalt einer DICOM-Studie verstehen, ohne die Bilddaten laden zu müssen. Bilder werden dank fortschrittlicher Bildkomprimierung schneller geladen, ohne Kompromisse bei der Bildqualität einzugehen. Die progressive Auflösung ermöglicht ein noch schnelleres Laden von Bildern für Miniaturansichten, interessante Bereiche und Mobilgeräte mit niedriger Auflösung.

Skalierbare DICOM-Importe

HealthImaging AWS-Importe nutzen moderne Cloud-native Technologien, um mehrere DICOM-Studien parallel zu importieren. Historische Archive können schnell importiert werden, ohne die klinische Arbeitslast für neue Daten zu beeinträchtigen. Informationen zu unterstützten SOP-Instanzen und Übertragungssyntaxen finden Sie unter. DICOM

Vom Service verwaltete DICOM-Datenhierarchie

AWS organisiert DICOM P10-Daten beim Import HealthImaging automatisch nach DICOM-Datenelementen auf Patienten-, Studien- und Serienebene. Der Service organisiert diese

DICOM-Daten in Bilddatensätzen, die der DICOM-Serie entsprechen, und vereinfacht so die Arbeitsabläufe nach dem Import. Die Organisation auf Studien- und Serienebene wird beibehalten, wenn neue Daten importiert werden.

DICOMweb API-Kompatibilität

AWS HealthImaging bietet DICOMweb konforme Lösungen, um Integrationen APIs zu vereinfachen und die Interoperabilität mit bestehenden Anwendungen zu ermöglichen. Der Service bietet auch Cloud-native Funktionen, die Aktionen ermöglichen APIs , die vom DICOMweb Standard nicht unterstützt werden, z. B. Operationen zur Aktualisierung von Metadaten.

Verwandte Dienste AWS

AWS HealthImaging bietet eine enge Integration mit anderen AWS Services. Kenntnisse der folgenden Services sind hilfreich, um sie in vollem Umfang nutzen zu können HealthImaging.

- AWS Identity and Access Management— Verwenden Sie IAM, um Identitäten und den Zugriff auf Ressourcen sicher zu HealthImaging verwalten.
- Amazon Simple Storage Service — Verwenden Sie Amazon S3 als Staging-Bereich für den Import von DICOM-Daten in. HealthImaging
- Amazon CloudWatch — Wird CloudWatch zur Beobachtung und Überwachung von HealthImaging Ressourcen verwendet.
- AWS CloudTrail— Wird verwendet CloudTrail , um HealthImaging Benutzeraktivitäten und API-Nutzung zu verfolgen.
- AWS CloudFormation— Wird verwendet CloudFormation , um IaC-Vorlagen (Infrastructure as Code) zu implementieren, in HealthImaging denen Ressourcen erstellt werden.
- AWS PrivateLink— Verwenden Sie Amazon VPC, um Konnektivität zwischen HealthImaging und Amazon Virtual Private Cloud herzustellen, ohne Daten dem Internet zugänglich zu machen.
- Amazon EventBridge — Wird verwendet, EventBridge um skalierbare, ereignisgesteuerte Anwendungen zu erstellen, indem Regeln erstellt werden, die HealthImaging Ereignisse an Ziele weiterleiten.

Zugreifen auf AWS HealthImaging

Sie können HealthImaging mit dem AWS-Managementkonsole, AWS Command Line Interface und dem auf AWS zugreifen AWS SDKs. Dieses Handbuch enthält Verfahrensanweisungen für das AWS-Managementkonsole und sowie Codebeispiele für AWS CLI und AWS SDKs.

AWS-Managementkonsole

Das AWS-Managementkonsole bietet eine webbasierte Benutzeroberfläche für die Verwaltung HealthImaging und die zugehörigen Ressourcen. Wenn Sie sich für ein AWS Konto registriert haben, können Sie sich an der HealthImaging Konsole anmelden.

AWS Command Line Interface (AWS CLI)

Das AWS CLI bietet Befehle für eine Vielzahl von AWS Produkten und wird unter Windows, Mac und Linux unterstützt. Weitere Informationen finden Sie im AWS Command Line Interface - Benutzerhandbuch.

AWS SDKs

AWS SDKs stellt Bibliotheken, Codebeispiele und andere Ressourcen für Softwareentwickler bereit. Diese Bibliotheken bieten grundlegende Funktionen, mit denen Aufgaben wie das kryptografische Signieren Ihrer Anfragen, das Wiederholen von Anfragen und das Behandeln von Fehlerantworten automatisiert werden können. Weitere Informationen finden Sie unter Tools, auf denen Sie aufbauen können. AWS

HTTP-Anforderungen

Sie können HealthImaging Aktionen mithilfe von HTTP-Anfragen aufrufen, müssen jedoch je nach Art der verwendeten Aktionen unterschiedliche Endpunkte angeben. Weitere Informationen finden Sie unter Unterstützte API-Aktionen für HTTP-Anfragen.

HIPAA-Eignung und Datensicherheit

Dies ist ein HIPAA-berechtigter Service. Weitere Informationen AWS zum US-amerikanischen Health Insurance Portability and Accountability Act von 1996 (HIPAA) und zur Nutzung von AWS Diensten zur Verarbeitung, Speicherung und Übertragung geschützter Gesundheitsinformationen (PHI) finden Sie unter HIPAA Overview.

Verbindungen, die PHI und personenbezogene Daten (PII) HealthImaging enthalten, müssen verschlüsselt sein. Standardmäßig HealthImaging verwenden alle Verbindungen HTTPS über TLS.

HealthImaging speichert verschlüsselte Kundeninhalte und arbeitet nach dem Modell der AWS gemeinsamen Verantwortung.

Informationen zur Einhaltung von Vorschriften finden Sie unterKonformitätsvalidierung für AWS HealthImaging.

Preisgestaltung

HealthImaging hilft Ihnen, das Lebenszyklusmanagement klinischer Daten mit intelligentem Tiering zu automatisieren. Weitere Informationen finden Sie unter Kostenoptimierung.

Allgemeine Preisinformationen finden Sie unter HealthImaging AWS-Preise. Verwenden Sie den HealthImaging AWS-Preisrechner, um die Kosten zu schätzen.

Erste Schritte mit AWS HealthImaging

Um mit der Nutzung von AWS zu beginnen HealthImaging, richten Sie ein AWS Konto ein und erstellen Sie einen AWS Identity and Access Management Benutzer. Um die AWS CLIoder die verwenden zu können AWS SDKs, müssen Sie sie installieren und konfigurieren.

Nachdem Sie sich mit den HealthImaging Konzepten und der Einrichtung vertraut gemacht haben, steht Ihnen ein kurzes Tutorial mit Codebeispielen zur Verfügung, das Ihnen den Einstieg erleichtern soll.

Themen

- HealthImaging AWS-Konzepte
- Einrichtung von AWS HealthImaging
- HealthImaging AWS-Tutorial

HealthImaging AWS-Konzepte

Die folgenden Begriffe und Konzepte sind für Ihr Verständnis und Ihre Verwendung von AWS von zentraler Bedeutung HealthImaging.

Konzepte

- Datastore
- Bilddatensatz
- Metadaten
- Bildrahmen

Datastore

Ein Datenspeicher ist ein Speicher für medizinische Bildgebungsdaten, der sich in einem einzigen AWS-Region Speicher befindet. Ein AWS Konto kann null oder viele Datenspeicher haben. Ein Datenspeicher hat seinen eigenen AWS KMS Verschlüsselungsschlüssel, sodass Daten in einem Datenspeicher physisch und logisch von Daten in anderen Datenspeichern isoliert werden können. Datenspeicher unterstützen die Zugriffskontrolle mithilfe von IAM-Rollen, Berechtigungen und attributbasierter Zugriffskontrolle.

Weitere Informationen erhalten Sie unter Verwaltung von Datenspeichern und Kostenoptimierung.

Bilddatensatz

Ein Bilddatensatz ist ein AWS Konzept, das einen abstrakten Gruppierungsmechanismus zur Optimierung verwandter medizinischer Bilddaten definiert. Wenn Sie Ihre DICOM P10-Bilddaten in einen HealthImaging AWS-Datenspeicher importieren, werden sie in Bildsätze umgewandelt, die aus Metadaten und Bildrahmen (Pixeldaten) bestehen. HealthImaging versucht, importierte Daten gemäß der DICOM-Hierarchie von Studie, Serie und Instanz zu organisieren. DICOM-Instanzen, die erfolgreich zur HealthImaging verwalteten Hierarchie hinzugefügt wurden, werden als primäre Bilddatensätze bezeichnet. Beim Import von DICOM P10-Daten wird entweder: ein neuer primärer Bildsatz erstellt, Instanzen zu einem vorhandenen primären Imagesatz zusammengeführt, falls die Instanzen bereits in der primären Sammlung vorhanden sind, oder, im Fall von Konflikten mit Metadatenelementen, ein neuer, nicht primärer Imagesatz erstellt.

Weitere Informationen erhalten Sie unter Imaging-Daten importieren und Bilddatensätze verstehen.

Metadaten

Metadaten sind die Nicht-Pixel-Attribute, die in einem Bilddatensatz vorhanden sind. Für DICOM gehören dazu demografische Daten des Patienten, Einzelheiten zum Verfahren und andere akquisitionsspezifische Parameter. AWS HealthImaging unterteilt den Bildsatz in Metadaten und Bildrahmen (Pixeldaten), sodass Anwendungen schnell darauf zugreifen können. Dies ist hilfreich für Bildbetrachter, Analysen und AI/ML Anwendungsfälle, für die keine Pixeldaten erforderlich sind. DICOM-Daten werden auf Patienten-, Studien- und Serienebene normalisiert, wodurch Inkonsistenzen vermieden werden. Dies vereinfacht die Verwendung der Daten, erhöht die Sicherheit und verbessert die Zugriffsleistung.

Weitere Informationen erhalten Sie unter Metadaten von Bilddatensätzen abrufen und Normalisierung von Metadaten.

Bildrahmen

Ein Bildrahmen sind die Pixeldaten, die in einem Bilddatensatz vorhanden sind, um ein medizinisches 2D-Bild zu erstellen. Einige Dateien behalten beim Import ihre ursprüngliche Übertragungssyntaxkodierung bei, während andere transkodiert werden. Amazon Web Services Services-Datenspeicher können so konfiguriert werden, dass sie verlustfreie Bildframes entweder in High-Throughput JPEG 2000 (HTJ2K) Lossless oder JPEG 2000 Lossless transkodieren. Wenn ein

Bildrahmen in HTJ2 K oder JPEG 2000 codiert ist, muss er vor der Anzeige in einem Bildbetrachter dekodiert werden. Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen, Pixeldaten des Bildsatzes abrufen und Bibliotheken zur Dekodierung von Bildrahmen.

Einrichtung von AWS HealthImaging

Sie müssen Ihre AWS Umgebung einrichten, bevor Sie AWS verwenden können HealthImaging. Die folgenden Themen sind Voraussetzungen für das Tutorial im nächsten Abschnitt.

Topics

- Melde dich an für eine AWS-Konto
- Erstellen Sie S3-Buckets
- Einen Datenspeicher erstellen
- Erstellen Sie einen IAM-Benutzer mit HealthImaging voller Zugriffsberechtigung
- Erstellen Sie eine IAM-Rolle für den Import
- Installieren Sie das AWS CLI (Optional)

Melde dich an für eine AWS-Konto

Um loszulegen AWS, benötigen Sie eine AWS-Konto. Informationen zum Erstellen eines AWS-Konto finden Sie unter Erste Schritte mit einem AWS-Konto im AWS -Kontenverwaltung Referenzhandbuch.

Erstellen Sie S3-Buckets

Für den Import von DICOM P10-Daten in AWS HealthImaging werden zwei Amazon S3 S3-Buckets empfohlen. Der Amazon S3 S3-Eingabe-Bucket speichert die zu importierenden DICOM P10-Daten und HealthImaging liest aus diesem Bucket. Der Amazon S3 S3-Ausgabe-Bucket speichert die Verarbeitungsergebnisse des Importjobs und HealthImaging schreibt in diesen Bucket. Eine visuelle Darstellung davon finden Sie im Diagramm unterImportaufträge verstehen.

> **Note**
>
> Aufgrund der AWS Identity and Access Management (IAM-) Richtlinie müssen Ihre Amazon S3 S3-Bucket-Namen eindeutig sein. Weitere Informationen finden Sie unter Benennungsregeln für Buckets im Benutzerhandbuch zu Amazon Simple Storage Service.

Für die Zwecke dieses Handbuchs spezifizieren wir die folgenden Amazon S3 S3-Eingabe- und Ausgabe-Buckets in der IAM-Rolle für den Import.

- Eingabe-Bucket: `arn:aws:s3:::amzn-s3-demo-source-bucket`
- Ausgabe-Bucket: `arn:aws:s3:::amzn-s3-demo-logging-bucket`

Weitere Informationen finden Sie unter Erstellen eines Buckets im Amazon S3 S3-Benutzerhandbuch.

Einen Datenspeicher erstellen

Wenn Sie Ihre medizinischen Bilddaten importieren, enthält der HealthImaging AWS-Datenspeicher die Ergebnisse Ihrer transformierten DICOM P10-Dateien, die als Bilddatensätze bezeichnet werden. Eine visuelle Darstellung davon finden Sie im Diagramm unter. Importaufträge verstehen

> **Tip**
>
> A `datastoreID` wird generiert, wenn Sie einen Datenspeicher erstellen. Sie müssen das verwenden`datastoreID`, wenn Sie das trust relationship für den Import später in diesem Abschnitt abschließen.

Informationen zum Erstellen eines Datenspeichers finden Sie unterEinen Datenspeicher erstellen.

Erstellen Sie einen IAM-Benutzer mit HealthImaging voller Zugriffsberechtigung

> **Bewährte Methode**
>
> Wir empfehlen Ihnen, separate IAM-Benutzer für unterschiedliche Anforderungen wie Import, Datenzugriff und Datenverwaltung zu erstellen. Dies steht im Einklang mit Grant Least Privilege Access im AWS Well-Architected Framework.
> Für das Tutorial im nächsten Abschnitt verwenden Sie einen einzelnen IAM-Benutzer.

So erstellen Sie einen IAM-Benutzer

1. Folgen Sie den Anweisungen zum Erstellen eines IAM-Benutzers in Ihrem AWS Konto im IAM-Benutzerhandbuch. Erwägen Sie zur Verdeutlichung die Benennung des Benutzers `ahiadmin` (oder etwas Ähnliches).
2. Weisen Sie dem IAM-Benutzer die `AWSHealthImagingFullAccess` verwaltete Richtlinie zu. Weitere Informationen finden Sie unter AWS verwaltete Richtlinie: AWSHealthImagingFullAccess.

> **Note**
>
> IAM-Berechtigungen können eingeschränkt werden. Weitere Informationen finden Sie unter AWS verwaltete Richtlinien für AWS HealthImaging.

Erstellen Sie eine IAM-Rolle für den Import

> **Note**
>
> Die folgenden Anweisungen beziehen sich auf eine AWS Identity and Access Management (IAM-) Rolle, die Lese- und Schreibzugriff auf Amazon S3 S3-Buckets für den Import Ihrer DICOM-Daten gewährt. Obwohl die Rolle für das Tutorial im nächsten Abschnitt erforderlich ist, empfehlen wir Ihnen, Benutzern, Gruppen und Rollen, die sie verwenden, IAM-Berechtigungen hinzuzufügen, da sie einfacher zu verwenden sindAWS verwaltete Richtlinien für AWS HealthImaging, als selbst Richtlinien zu schreiben.

Eine IAM-Rolle ist eine IAM-Identität, die Sie in Ihrem Konto mit bestimmten Berechtigungen erstellen können. Um einen Importauftrag zu starten, muss die IAM-Rolle, die die `StartDICOMImportJob` Aktion aufruft, an eine Benutzerrichtlinie angehängt werden, die Zugriff auf die Amazon S3 S3-Buckets gewährt, die zum Lesen Ihrer DICOM P10-Daten und zum Speichern der Verarbeitungsergebnisse des Importauftrags verwendet werden. Ihm muss auch eine Vertrauensbeziehung (Richtlinie) zugewiesen werden, die es AWS ermöglicht, die Rolle HealthImaging zu übernehmen.

Um eine IAM-Rolle für Importzwecke zu erstellen

1. Erstellen Sie mithilfe der IAM-Konsole eine Rolle mit dem Namen. `ImportJobDataAccessRole` Sie verwenden diese Rolle für das Tutorial im nächsten Abschnitt. Weitere Informationen finden Sie unter Erstellen von IAM-Rollen im -IAM-Benutzerhandbuch.

   > **Tip**
   >
   > Für die Zwecke dieses Handbuchs beziehen sich die Einen Importjob starten Codebeispiele auf die `ImportJobDataAccessRole` IAM-Rolle.

2. Fügen Sie der IAM-Rolle eine IAM-Berechtigungsrichtlinie hinzu. Diese Berechtigungsrichtlinie gewährt Zugriff auf die Amazon S3 S3-Eingabe- und Ausgabe-Buckets. Fügen Sie der IAM-Rolle die folgende Berechtigungsrichtlinie hinzu. `ImportJobDataAccessRole`

   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. Fügen Sie der `ImportJobDataAccessRole` IAM-Rolle die folgende Vertrauensbeziehung (Richtlinie) hinzu. Die Vertrauensrichtlinie erfordert die`datastoreId`, die generiert wurde, als Sie den Abschnitt Einen Datenspeicher erstellen abgeschlossen haben. In der Anleitung zu diesem Thema wird davon ausgegangen, dass Sie einen HealthImaging AWS-Datenspeicher verwenden, jedoch mit datenspeicherspezifischen Amazon S3 S3-Buckets, IAM-Rollen und Vertrauensrichtlinien.

> **Note**
>
> Der `Condition` Block in dieser Vertrauensrichtlinie trägt dazu bei, das Problem des verwirrten Stellvertreters zu vermeiden, indem sichergestellt wird, dass nur auf Ihren spezifischen HealthImaging AWS-Datenspeicher zugegriffen werden kann. Weitere Informationen zu dieser Sicherheitsmaßnahme finden Sie unter Cross-service Confused Deputy Prevention in HealthImaging.

JSON

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

```
  ]
}
```

Weitere Informationen zur Erstellung und Verwendung von IAM-Richtlinien mit AWS finden Sie HealthImaging unterIdentity and Access Management für AWS HealthImaging.

Weitere Informationen zu IAM-Rollen im Allgemeinen finden Sie unter IAM-Rollen im IAM-Benutzerhandbuch. Weitere Informationen zu IAM-Richtlinien und -Berechtigungen im Allgemeinen finden Sie unter IAM-Richtlinien und -Berechtigungen im IAM-Benutzerhandbuch.

Installieren Sie das AWS CLI (Optional)

Das folgende Verfahren ist erforderlich, wenn Sie das verwenden AWS Command Line Interface. Wenn Sie die AWS SDKs AWS-Managementkonsole oder verwenden, können Sie das folgende Verfahren überspringen.

Um das einzurichten AWS CLI

1. Herunterladen und Konfigurieren von AWS CLI. Eine Anleitung finden Sie unter den folgenden Themen im AWS Command Line Interface -Benutzerhandbuch.

   - Installation oder Aktualisierung der neuesten Version von AWS CLI
   - Erste Schritte mit dem AWS CLI

2. Fügen Sie der AWS CLI `config` Datei ein benanntes Profil für den Administrator hinzu. Sie verwenden dieses Profil, wenn Sie die AWS CLI Befehle ausführen. Gemäß dem Sicherheitsprinzip der geringsten Rechte empfehlen wir Ihnen, eine separate IAM-Rolle mit spezifischen Rechten für die auszuführenden Aufgaben zu erstellen. Weitere Informationen zu benannten Profilen finden Sie im AWS Command Line Interface Benutzerhandbuch unter Konfiguration und Einstellungen für Anmeldeinformationsdateien.

   ```
   [default]
   aws_access_key_id = default access key ID
   aws_secret_access_key = default secret access key
   region = region
   ```

3. Überprüfen Sie das Setup mit dem folgenden `help` Befehl.

   ```
   aws medical-imaging help
   ```

Wenn der richtig konfiguriert AWS CLI ist, sehen Sie eine kurze Beschreibung von AWS HealthImaging und eine Liste der verfügbaren Befehle.

HealthImaging AWS-Tutorial

Ziel

Das Ziel dieses Tutorials besteht darin, DICOM P10-Binärdateien (`.dcm`Dateien) in einen HealthImaging AWS-Datenspeicher zu importieren und sie in Bildsätze umzuwandeln, die aus Metadaten und Bildrahmen (Pixeldaten) bestehen. Nach dem Import der DICOM-Daten verwenden Sie HealthImaging Cloud-native Aktionen, um je nach Ihren Zugriffspräferenzen auf die Bildsätze, Metadaten und Bildrahmen zuzugreifen.

Voraussetzungen

Alle unter aufgeführten Verfahren Einrichtung sind erforderlich, um dieses Tutorial abzuschließen.

Schritte des Tutorials

1. Starten Sie den Importjob
2. Rufen Sie die Eigenschaften des Importauftrags ab
3. Suchen Sie nach Bilddatensätzen
4. Holen Sie sich die Eigenschaften von Bildsätzen
5. Holen Sie sich Metadaten von Bilddatensätzen
6. Holen Sie sich Pixeldaten des Bildsatzes
7. Datenspeicher löschen

Verwaltung von Datenspeichern mit AWS HealthImaging

Mit AWS HealthImaging erstellen und verwalten Sie Datenspeicher für medizinische Bildressourcen. In den folgenden Themen wird beschrieben, wie Sie HealthImaging Cloud-native Aktionen verwenden, um Datenspeicher mithilfe von, und zu erstellen, zu beschreiben AWS-Managementkonsole AWS CLI, aufzulisten und zu löschen AWS SDKs.

> **Note**
>
> Das letzte Thema in diesem Kapitel befasst sich mit der Kostenoptimierung. Nachdem Sie Ihre medizinischen Bilddaten in einen HealthImaging Datenspeicher importiert haben, werden sie je nach Zeit und Nutzung automatisch zwischen zwei Speicherebenen verschoben. Für die Speicherstufen gibt es unterschiedliche Preisstufen. Daher ist es wichtig, den Prozess der Stufenverschiebung und die HealthImaging Ressourcen zu verstehen, die für Abrechnungszwecke anerkannt werden.

Themen

- Einen Datenspeicher erstellen
- Eigenschaften des Datenspeichers abrufen
- Datenspeicher auflisten
- Löschen eines Datenspeichers

Einen Datenspeicher erstellen

Verwenden Sie die `CreateDatastore` Aktion, um einen HealthImaging AWS-Datenspeicher für den Import von DICOM P10-Dateien zu erstellen. Die folgenden Menüs enthalten ein Verfahren für das AWS-Managementkonsole und Codebeispiele für und. AWS CLI AWS SDKs Weitere Informationen finden Sie `CreateDatastore`in der HealthImaging AWS-API-Referenz. Wenn Sie einen Datenspeicher erstellen, können Sie die Standardübertragungssyntax auswählen, die AWS zum Transcodieren und Speichern von verlustfreien Bildframes HealthImaging verwendet hat. Diese Konfiguration kann nach der Erstellung des Datenspeichers nicht geändert werden.

JPEG 2000 (HTJ2K) mit hohem Durchsatz

HTJ2K (JPEG 2000 mit hohem Durchsatz) ist das Standardspeicherformat für HealthImaging Datenspeicher. Es ist eine Erweiterung des JPEG 2000-Standards, die eine deutlich verbesserte Kodierungs- und Dekodierungsleistung bietet. Wenn Sie einen Datenspeicher ohne Angabe von a erstellen`–lossless-storage-format`, HealthImaging wird automatisch K verwendet. HTJ2 Informationen zum Erstellen eines Datenspeichers mit HTJ2 K finden Sie in der AWS-CLI und im folgenden SDKs Abschnitt.

JPEG 2000 Lossless

Die JPEG 2000-Lossless-Kodierung ermöglicht die Erstellung von Datenspeichern, die persistent sind und die verlustfreien Bildframes im JPEG 2000-Format ohne Transcodierung abrufen. Dies ermöglicht eine kürzere Latenz beim Abrufen von Anwendungen, die JPEG 2000 Lossless (DICOM Transfer Syntax UID 1.2.840.10008.1.2.4.90) benötigen. Weitere Informationen finden Sie unter. Unterstützte Übertragungssyntaxen Informationen zum Erstellen eines Datenspeichers im verlustfreien Format JPEG 2000 finden Sie in der AWS-CLI und im nachfolgenden SDKs Abschnitt.

> **Wichtig**
>
> - Benennen Sie keine Datenspeicher mit geschützten Gesundheitsinformationen (PHI), personenbezogenen Daten (PII) oder anderen vertraulichen oder sensiblen Informationen.
> - Die AWS-Konsole unterstützt die Erstellung von Datenspeichern mit Standardeinstellungen. Verwenden Sie die AWS CLI oder AWS das SDK, um einen Datenspeicher mit einem `–lossless-storage-format` angegebenen optionalen Wert zu erstellen.

So erstellen Sie einen Datenspeicher

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher erstellen in der HealthImaging Konsole.
2. Geben Sie unter Details für Name des Datenspeichers einen Namen für Ihren Datenspeicher ein.
3. Wählen Sie unter Datenverschlüsselung einen AWS KMS Schlüssel für die Verschlüsselung Ihrer Ressourcen aus. Weitere Informationen finden Sie unter Datenschutz in AWS HealthImaging.

4. Unter Tags — optional können Sie Ihrem Datenspeicher bei der Erstellung Tags hinzufügen. Weitere Informationen finden Sie unter Eine Ressource taggen.
5. Wählen Sie Datenspeicher erstellen aus.

AWS CLI und SDKs

Bash

AWS CLI mit Bash-Skript

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

##############################################################################
# function imaging_create_datastore
#
# This function creates an AWS HealthImaging data store for importing DICOM P10
 files.
#
# Parameters:
#       -n data_store_name - The name of the data store.
#
# Returns:
#       The datastore ID.
#    And:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function imaging_create_datastore() {
  local datastore_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_create_datastore"
    echo "Creates an AWS HealthImaging data store for importing DICOM P10 files."
```

```
    echo "  -n data_store_name - The name of the data store."
    echo ""
  }

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

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

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

  local error_code=${?}

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

  echo "$response"

  return 0
```

```
}
```

- Einzelheiten zur API finden Sie CreateDatastorein der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: Um einen Datenspeicher zu erstellen

Im folgenden Codebeispiel für `create-datastore` wird ein Datenspeicher mit dem Namen `my-datastore` erstellt. Wenn Sie einen Datenspeicher ohne Angabe von a erstellen`--lossless-storage-format`, wird AWS HealthImaging standardmäßig HTJ2 K (High Throughput JPEG 2000) verwendet.

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

Ausgabe:

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

Beispiel 2: So erstellen Sie einen Datenspeicher mit dem verlustfreien JPEG 2000-Speicherformat

Ein Datenspeicher, der mit dem verlustfreien Speicherformat JPEG 2000 konfiguriert ist, transkodiert und speichert verlustfreie Bildframes im JPEG 2000-Format. Bildframes können dann ohne Transcodierung im Format JPEG 2000 Lossless abgerufen werden. Im folgenden `create-datastore` Codebeispiel wird ein für das verlustfreies JPEG 2000-Speicherformat konfigurierter Datenspeicher mit dem Namen `my-datastore` erstellt.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Erstellen eines Datenspeichers.

- Einzelheiten zur API finden Sie CreateDatastoreunter AWS CLI Befehlsreferenz.

Java

SDK für 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 "";
    }
```

- Einzelheiten zur API finden Sie CreateDatastorein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

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

- Einzelheiten zur API finden Sie CreateDatastorein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie CreateDatastorein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie CreateDatastorein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Eigenschaften des Datenspeichers abrufen

Verwenden Sie die `GetDatastore` Aktion, um die Eigenschaften des HealthImaging AWS-Datenspeichers abzurufen. Die folgenden Menüs enthalten ein Verfahren für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `GetDatastore`in der HealthImaging AWS-API-Referenz.

Um die Eigenschaften des Datenspeichers abzurufen

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Details zum Datenspeicher wird geöffnet. Im Abschnitt Details sind alle Eigenschaften des Datenspeichers verfügbar. Um die zugehörigen Bilddatensätze, Importe und Tags anzuzeigen, wählen Sie die entsprechende Registerkarte aus.

AWS CLI und SDKs

Bash

AWS CLI mit Bash-Skript

```
##############################################################################
```

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

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

  # Retrieve the calling parameters.
  while getopts "i:h" option; do
    case "${option}" in
      i) datastore_id="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
```

```
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

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

  local response

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

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

  echo "$response"

  return 0
}
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: Um die Eigenschaften eines Datenspeichers abzurufen

Im folgenden Codebeispiel für `get-datastore` werden die Eigenschaften eines Datenspeichers abgerufen.

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

Ausgabe:

```
{
    "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"
    }
}
```

Beispiel 2: Um die Eigenschaften eines Datenspeichers für JPEG2 000 zu konfigurieren

Im folgenden `get-datastore` Codebeispiel werden die Eigenschaften eines Datenspeichers für einen Datenspeicher abgerufen, der für das Format JPEG 2000 Lossless konfiguriert ist.

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

Ausgabe:

```
{
    "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"
    }
}
```

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Datenspeichereigenschaften.

- Einzelheiten zur API finden Sie GetDatastorein der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
```

```
};
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetDatastorein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    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.
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Datenspeicher auflisten

Verwenden Sie die `ListDatastores` Aktion, um verfügbare Datenspeicher in AWS aufzulisten HealthImaging. Die folgenden Menüs enthalten ein Verfahren für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `ListDatastores`in der HealthImaging AWS-API-Referenz.

So listen Sie Datenspeicher auf

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

- Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.

  Alle Datenspeicher sind im Abschnitt Datenspeicher aufgeführt.

AWS CLI und SDKs

Bash

AWS CLI mit Bash-Skript

```
###############################################################################
# function errecho
```

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

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

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

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

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

  echo "$response"

  return 0
}
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So listen Sie Datenspeicher auf

Im folgenden Codebeispiel für `list-datastores` werden die verfügbaren Datenspeicher aufgelistet.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Developer Guide unter Auflisten von Datenspeichern.

- Einzelheiten zur API finden Sie ListDatastoresin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
   }
```

- Einzelheiten zur API finden Sie ListDatastoresunter AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

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

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

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page.datastoreSummaries);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
```

```
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie ListDatastoresin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Löschen eines Datenspeichers

Verwenden Sie die `DeleteDatastore` Aktion, um einen HealthImaging AWS-Datenspeicher zu löschen. Die folgenden Menüs enthalten ein Verfahren für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `DeleteDatastore`in der HealthImaging AWS-API-Referenz.

> **Note**
>
> Bevor ein Datenspeicher gelöscht werden kann, müssen Sie zunächst alle darin enthaltenen Bildsätze löschen. Weitere Informationen finden Sie unter Löschen eines Bilddatensatzes.

So löschen Sie einen Datenspeicher

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.
3. Wählen Sie Löschen aus.

   Die Seite Datenspeicher löschen wird geöffnet.
4. Um das Löschen des Datenspeichers zu bestätigen, geben Sie den Namen des Datenspeichers in das Texteingabefeld ein.
5. Wählen Sie Datenspeicher löschen.

AWS CLI und SDKs

Bash

AWS CLI mit Bash-Skript

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

##############################################################################
# function imaging_delete_datastore
#
# This function deletes an AWS HealthImaging data store.
#
# Parameters:
#       -i datastore_id - The ID of the data store.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
```

```
################################################################################
function imaging_delete_datastore() {
  local datastore_id response
  local option OPTARG # Required to use getopts command in a function.

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

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

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

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

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports medical-imaging delete-datastore operation
 failed.$response"
    return 1
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So löschen Sie einen Datenspeicher

Im folgenden Codebeispiel für `delete-datastore` wird ein Datenspeicher gelöscht.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Löschen eines Datenspeichers.

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
       }
   }
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId }),
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //         },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie DeleteDatastorein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    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.
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Verwendung DICOMweb mit AWS HealthImaging

Sie können DICOM-Objekte HealthImaging mithilfe einer Darstellung von DICOMweb APIsaus AWS abrufen. Diese sind webbasiert und entsprechen APIs dem DICOM-Standard für medizinische Bildgebung. Diese Funktion ermöglicht es Ihnen, mit Systemen zusammenzuarbeiten, die DICOM Part 10-Binärdateien verwenden, und gleichzeitig die Vorteile der HealthImaging Cloud-nativen Aktionen zu nutzen. Der Schwerpunkt dieses Kapitels liegt auf der Verwendung der Implementierung HealthImaging von DICOMweb APIs to return response. DICOMweb

> **Wichtig**
>
> HealthImaging speichert DICOM-Daten als Bilddatensätze. Verwenden Sie HealthImaging Cloud-native Aktionen, um Bilddatensätze zu verwalten und abzurufen. HealthImaging's DICOMweb APIs kann verwendet werden, um Bilddatensatzinformationen mit DICOMweb -konformen Antworten zurückzugeben.
> Die in diesem Kapitel APIs aufgelisteten Daten entsprechen dem DICOMwebStandard für webbasierte medizinische Bildgebung. Da es sich um Repräsentationen von handelt DICOMweb APIs, werden sie nicht über AWS CLI und AWS SDKs angeboten.

Thema

- Speichern von Instanzen mit STOW-RS
- DICOM-Daten werden abgerufen von HealthImaging
- DICOM-Daten durchsuchen in HealthImaging
- OIDC-Authentifizierung für DICOMweb APIs

Speichern von Instanzen mit STOW-RS

AWS HealthImaging bietet eine Darstellung der `DICOMweb STOW-RS` APIs für den Import von Daten. Verwenden Sie diese APIs , um DICOM-Daten synchron in Ihrem HealthImaging Datenspeicher zu speichern.

In der folgenden Tabelle werden die HealthImaging Repräsentationen von DICOMweb STOW-RS beschrieben, die für den Import von Daten APIs verfügbar sind.

HealthImaging Darstellungen von STOW-RS DICOMweb APIs

Name	Description
`StoreDICOM`	Speichern Sie eine oder mehrere Instanzen in einem HealthImaging Datenspeicher.
`StoreDICOMStudy`	Speichern Sie eine oder mehrere Instanzen , die einer angegebenen Studieninstanz-UID entsprechen, in einem HealthImaging Datenspeicher.

Mit den `StoreDICOMStudy` Aktionen `StoreDICOM` und importierte Daten werden als neue primäre Bilddatensätze organisiert oder zu vorhandenen primären Bilddatensätzen hinzugefügt. Dabei wird dieselbe Logik wie bei asynchronen Importaufträgen verwendet. Wenn die Metadatenelemente neu importierter DICOM P10-Daten mit vorhandenen primären Bilddatensätzen kollidieren, werden die neuen Daten zu nicht primären Bilddatensätzen hinzugefügt.

> **Note**
>
> - Diese Aktionen unterstützen das Hochladen von bis zu 1 GB DICOM-Daten pro Anfrage.
> - Die API-Antwort wird im JSON-Format vorliegen und dem STOW-RS-Standard entsprechen. DICOMweb

Um eine StoredICOM-Anfrage zu initiieren

1. Erfassen Sie Ihre AWS-Region und den DICOM P10-Dateinamen. HealthImaging `datastoreId`
2. Konstruieren Sie eine URL für die Anfrage des Formulars: `https://dicom-medical-imaging.`*`region`*`.amazonaws.com/datastore/`*`datastore-id`*`/studies`
3. Ermitteln Sie beispielsweise die Inhaltslänge der DICOM P10-Datei mit Ihrem bevorzugten Befehl. `$(stat -f %z $FILENAME)`
4. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `StoreDICOM`verwendet eine HTTP-POST-Anforderung mit dem AWS Signature Version 4-Signaturprotokoll.

Example Beispiel 1: Um eine DICOM P10-Datei mithilfe der Aktion zu speichern **`StoreDICOM`**

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 Beispiel 2: Um eine DICOM P10-Datei mithilfe der Aktion zu speichern **`StoreDICOMStudy`**

Der einzige Unterschied zwischen StoreDICOM und Store besteht DICOMStudy darin, dass eine Study Instance UID als Parameter an Store DICOMStudy übergeben wird und die hochgeladenen Instanzen Mitglieder der angegebenen Studie sein müssen.

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 Beispiel 3: Um DICOM P10-Dateien mit einer mehrteiligen HTTP-Payload zu speichern

Mit einer einzigen mehrteiligen Upload-Aktion können mehrere P10-Dateien hochgeladen werden. Die folgenden Shell-Befehle veranschaulichen, wie Sie eine mehrteilige Payload mit zwei P10-Dateien zusammenstellen und zusammen mit der Aktion hochladen. `StoreDICOM`

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

## Obain 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
```

DICOM-Daten werden abgerufen von HealthImaging

AWS HealthImaging bietet Darstellungen DICOMweb WADO-RS APIs zum Abrufen von Daten auf Serien- und Instanzebene. Mit diesen ist es möglich APIs, alle Metadaten für eine DICOM-Serie aus einem HealthImaging Datenspeicher abzurufen. Es ist auch möglich, eine DICOM-Instanz, DICOM-Instanz-Metadaten und DICOM-Instanz-Frames (Pixeldaten) abzurufen. HealthImagingDICOMweb WADO-RS APIs bietet Flexibilität beim Abrufen von Daten, die in älteren Anwendungen gespeichert sind, HealthImaging und sorgen für Interoperabilität mit älteren Anwendungen.

> **Wichtig**
>
> HealthImaging speichert DICOM-Daten als Bilddatensätze. Verwenden Sie HealthImaging Cloud-native Aktionen, um Bilddatensätze zu verwalten und abzurufen. HealthImaging's DICOMweb APIs kann verwendet werden, um Bilddatensatzinformationen mit DICOMweb -konformen Antworten zurückzugeben.
> Die in diesem Abschnitt APIs aufgelisteten Daten entsprechen dem DICOMweb (WADO-RS) -Standard für webbasierte medizinische Bildgebung. Da es sich um Repräsentationen von handelt DICOMweb APIs, werden sie nicht über und angeboten. AWS CLI AWS SDKs

In der folgenden Tabelle werden alle HealthImaging Repräsentationen von DICOMweb WADO-RS beschrieben, aus APIs denen Daten abgerufen werden können. HealthImaging

HealthImaging Repräsentationen von WADO-RS DICOMweb APIs

Name	Description
GetDICOMSeriesMetadata	Rufen Sie DICOM-Instanz-Metadaten (.jsonDatei) für eine DICOM-Serie in einem HealthImaging Datenspeicher ab, indem Sie die Studie und die Serie angeben, die einer Ressource UIDs zugeordnet sind. Siehe Metadaten der Serie abrufen.
GetDICOMInstance	Rufen Sie eine DICOM-Instanz (.dcmDatei) aus einem HealthImaging Datenspeicher

Name	Description
	ab, indem Sie die mit einer Ressource UIDs verknüpfte Serie, Studie und Instanz angeben. Siehe Rufen Sie eine Instanz ab.
`GetDICOMInstanceMetadata`	Rufen Sie DICOM-Instanz-Metadaten (`.json`Datei) aus einer DICOM-Instanz in einem HealthImaging Datenspeicher ab, indem Sie die mit einer Ressource UIDs verknüpft e Serie, Studie und Instanz angeben. Siehe Abrufen von Instance-Metadaten.
`GetDICOMInstanceFrames`	Rufen Sie einzelne oder Batch-Image-Frames (`multipart` Anfrage) von einer DICOM-Ins tanz in einem HealthImaging Datenspeicher ab, indem Sie die Serien-UID, die Studien-UID, die Instanz und die Frame-Nummern angeben UIDs, die einer Ressource zugeordnet sind. Siehe Frames abrufen.

Themen

- Abrufen einer DICOM-Instanz von HealthImaging
- Metadaten der DICOM-Instanz abrufen von HealthImaging
- Metadaten der DICOM-Serie abrufen von HealthImaging
- DICOM-Instanz-Frames abrufen von HealthImaging
- DICOM-Massendaten abrufen von HealthImaging

Abrufen einer DICOM-Instanz von HealthImaging

Verwenden Sie die `GetDICOMInstance` Aktion, um eine DICOM-Instanz (`.dcm`Datei) aus einem HealthImaging Datenspeicher abzurufen, indem Sie die Serie, Studie und Instanz angeben, die der UIDs Ressource zugeordnet sind. Die API gibt nur Instanzen aus primären Bilddatensätzen zurück, sofern der optionale Bilddatensatzparameter nicht angegeben wird. Sie können jede Instanz (aus primären oder nicht primären Bilddatensätzen) im Datenspeicher abrufen, indem Sie den

`imageSetId` als Abfrageparameter angeben. DICOM-Daten können entweder in der gespeicherten Übertragungssyntax oder im unkomprimierten Format (ELE) abgerufen werden.

Um eine DICOM-Instanz zu erhalten (`.dcm`)

1. Werte sammeln HealthImaging `datastoreId` und `imageSetId` parametrieren.
2. Verwenden Sie die `GetImageSetMetadata`Aktion mit den `imageSetId` Parameterwerten `datastoreId` und, um die zugehörigen Metadatenwerte für `studyInstanceUID`, `seriesInstanceUID` und abzurufen`sopInstanceUID`. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen abrufen.
3. Konstruieren Sie mithilfe der Werte für`datastoreId`,, `studyInstanceUID` `seriesInstanceUID`, `sopInstanceUID` und eine URL für die Anforderung`imageSetId`. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL-Pfad im folgenden Beispiel anzuzeigen. Die URL hat die Form:

   ```
   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. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `GetDICOMInstance`verwendet eine HTTP-GET-Anfrage mit dem AWS Signaturprotokoll Signature Version 4. Das folgende Codebeispiel verwendet das `curl` Befehlszeilentool, um eine DICOM-Instanz (`.dcm`Datei) von HealthImaging abzurufen.

   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**
>
> Die `transfer-syntax` UID ist optional und wird standardmäßig auf Explicit VR Little Endian gesetzt, falls sie nicht enthalten ist. Zu den unterstützten Übertragungssyntaxen gehören:
>
> - Explicit VR Little Endian (ELE) — `1.2.840.10008.1.2.1` (Standard für verlustfreie Bildrahmen)
> - `transfer-syntax=*`Wenn ja, werden die Bildrahmen in der gespeicherten Übertragungssyntax zurückgegeben.
> - JPEG 2000 mit hohem Durchsatz und RPCL-Optionen Bildkomprimierung (nur verlustfrei) — `1.2.840.10008.1.2.4.202` — wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.202`
> - JPEG 2000 Lossless — `1.2.840.10008.1.2.4.90` — wenn die Instanz als Lossless gespeichert ist. HealthImaging
> - JPEG-Baseline (Prozess 1): Standardübertragungssyntax für die verlustbehaftete 8-Bit-JPEG-Bildkomprimierung `1.2.840.10008.1.2.4.50` —, wenn die Instanz gespeichert ist in HealthImaging `1.2.840.10008.1.2.4.50`
> - JPEG 2000-Bildkomprimierung — `1.2.840.10008.1.2.4.91` — wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.91`
> - JPEG 2000-Bildkomprimierung mit hohem Durchsatz `1.2.840.10008.1.2.4.203` - - wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.203`
> - JPEG XL-Bildkomprimierung `1.2.840.10008.1.2.4.112` - - wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.112`
> - Instanzen, in HealthImaging denen ein oder mehrere Bildrahmen gespeichert sind, die in der MPEG-Familie von Transfersyntaxen (zu der auch MPEG-4 AVC/H.264 and HEVC/H .265 gehören) kodiert sind MPEG2, können mit der entsprechenden Transfer-Syntax-UID abgerufen werden. Zum Beispiel, `1.2.840.10008.1.2.4.100` wenn die Instanz als Main Profile Main Level gespeichert ist. MPEG2

Weitere Informationen erhalten Sie unter Unterstützte Übertragungssyntaxen und Bibliotheken zur Dekodierung von Image-Frames für AWS HealthImaging.

Metadaten der DICOM-Instanz abrufen von HealthImaging

Verwenden Sie die `GetDICOMInstanceMetadata` Aktion, um die Metadaten aus einer DICOM-Instanz in einem HealthImaging Datenspeicher abzurufen, indem Sie die Serie, Studie und Instanz angeben, die der UIDs Ressource zugeordnet sind. Die API gibt nur Instanz-Metadaten aus primären Bilddatensätzen zurück, sofern der optionale Bilddatensatzparameter nicht angegeben wird. Sie können alle Instanz-Metadaten (aus primären oder nicht primären Bilddatensätzen) im Datenspeicher abrufen, indem Sie den `imageSetId` als Abfrageparameter angeben.

Um DICOM-Instanz-Metadaten abzurufen () **`.json`**

1. Werte sammeln HealthImaging `datastoreId` und `imageSetId` parametrieren.
2. Verwenden Sie die `GetImageSetMetadata`Aktion mit den `imageSetId` Parameterwerten `datastoreId` und, um die zugehörigen Metadatenwerte für `studyInstanceUIDseriesInstanceUID`, und abzurufen`sopInstanceUID`. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen abrufen.
3. Konstruieren Sie mithilfe der Werte für`datastoreId`, `studyInstanceUID` `seriesInstanceUIDsopInstanceUID`, und eine URL für die Anforderung`imageSetId`. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL-Pfad im folgenden Beispiel anzuzeigen. Die URL hat die Form:

   ```
   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. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `GetDICOMInstanceMetadata`verwendet eine HTTP-GET-Anfrage mit dem AWS Signaturprotokoll Signature Version 4. Das folgende Codebeispiel verwendet das `curl` Befehlszeilentool, um Metadaten (`.json`Datei) der DICOM-Instanz HealthImaging abzurufen.

   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**
>
> Die in den Metadaten angegebene UID für die Übertragungssyntax entspricht der UID () `StoredTransferSyntaxUID` für die gespeicherte Übertragungssyntax in. HealthImaging

Metadaten der DICOM-Serie abrufen von HealthImaging

Verwenden Sie die `GetDICOMSeriesMetadata` Aktion, um die Metadaten für eine DICOM-Serie (`.json`Datei) aus einem HealthImaging Datenspeicher abzurufen. Sie können Serienmetadaten für jeden primären Bilddatensatz im HealthImaging Datenspeicher abrufen, indem Sie die Studie und die Serie angeben, die der Ressource UIDs zugeordnet sind. Sie können Serienmetadaten für Bilddatensätze abrufen, die nicht primär sind, indem Sie die Bilddatensatz-ID als Abfrageparameter angeben. Die Serienmetadaten werden im `DICOM JSON` Format zurückgegeben.

Um Metadaten der DICOM-Serie abzurufen () **`.json`**

1. Werte sammeln HealthImaging `datastoreId` und `imageSetId` parametrieren.
2. Konstruieren Sie eine URL für die Anfrage, indem Sie die Werte für `datastoreIdstudyInstanceUID`,`seriesInstanceUID`, und optional verwenden`imageSetId`. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL-Pfad im folgenden Beispiel anzuzeigen. Die URL hat die Form:

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

3. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `GetDICOMSeriesMetadata`verwendet eine HTTP-GET-Anfrage mit dem AWS Signaturprotokoll Signature Version 4. Das folgende Codebeispiel verwendet das `curl` Befehlszeilentool, um Metadaten (`.json`Datei) von abzurufen HealthImaging.

Shell

```
curl --request GET \
 'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
d9a2a515ab294163a2d2f4069eed584c/
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/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'
```

Mit dem optionalen `imageSetId` Parameter.

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

- Der `imageSetId` Parameter ist erforderlich, um Serienmetadaten für nicht primäre Bilddatensätze abzurufen. Die `GetDICOMInstanceMetadata` Aktion gibt nur Serienmetadaten für primäre Bilddatensätze zurück, wenn die`datastoreId`,`studyInstanceUID`, angegeben `seriesInstanceUID` sind (ohne ein`imagesetID`).

DICOM-Instanz-Frames abrufen von HealthImaging

Verwenden Sie die `GetDICOMInstanceFrames` Aktion, um einzelne Bild-Frames oder Batch-Frames (`multipart`Anfrage) von einer DICOM-Instance in einem HealthImaging Datenspeicher abzurufen, indem Sie die Serien-UID, die Studien-UID, die Instanz und die Frame-Nummern angeben UIDs, die einer Ressource zugeordnet sind. Sie können den Bilddatensatz angeben, aus dem Instanz-Frames abgerufen werden sollen, indem Sie die Bilddatensatz-ID als Abfrageparameter angeben. Die API gibt nur Instanzframes aus primären Bilddatensätzen zurück, es sei denn, der optionale Bildsatzparameter wird bereitgestellt. Sie können jeden Instanz-Frame (aus primären oder nicht primären Bilddatensätzen) im Datenspeicher abrufen, indem Sie den `imageSetId` als Abfrageparameter angeben.

DICOM-Daten können entweder in der gespeicherten Übertragungssyntax oder im unkomprimierten Format (ELE) abgerufen werden.

Um DICOM-Instanz-Frames () abzurufen **`multipart`**

1. Werte sammeln HealthImaging `datastoreId` und `imageSetId` parametrieren.
2. Verwenden Sie die `GetImageSetMetadata`Aktion mit den `imageSetId` Parameterwerten `datastoreId` und, um die zugehörigen Metadatenwerte für `studyInstanceUIDseriesInstanceUID`, und abzurufen`sopInstanceUID`. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen abrufen.
3. Ermitteln Sie die Bildrahmen, die aus den zugehörigen Metadaten abgerufen werden sollen, um den `frameList` Parameter zu bilden. Der `frameList` Parameter ist eine durch Kommas getrennte Liste mit einer oder mehreren Frame-Nummern, die nicht doppelt vorhanden sind, in beliebiger Reihenfolge. Der erste Bildrahmen in den Metadaten ist beispielsweise Frame 1.
   - Einzelrahmen-Anfrage: `/frames/1`
   - Anfrage mit mehreren Frames: `/frames/1,2,3,4`
4. Konstruieren Sie eine URL für die Anfrage mit den Werten für `datastoreIdstudyInstanceUID`,`seriesInstanceUID`, `sopInstanceUIDimageSetId`, und`frameList`. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL-Pfad im folgenden Beispiel anzuzeigen. Die URL hat die Form:

```
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. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `GetDICOMInstanceFrames`verwendet eine HTTP-GET-Anfrage mit dem AWS Signaturprotokoll Signature Version 4. Das folgende Codebeispiel verwendet das `curl` Befehlszeilentool, um Bildrahmen in einer `multipart` Antwort von abzurufen HealthImaging.

   Shell

```
curl --request GET \
  'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
d9a2a515ab294163a2d2f4069eed584c/
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/
instances/1.3.6.1.4.1.5962.1.1.4.1.1.20040826186059.5457/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**
>
> Die `transfer-syntax` UID ist optional und wird standardmäßig auf Explicit VR Little Endian gesetzt, falls sie nicht enthalten ist. Wenn die Transcodierung nach ELE nicht möglich ist (aufgrund eines Imports mit Warnung), werden Pixel ohne Transcodierung zurückgegeben. Zu den unterstützten Übertragungssyntaxen gehören:
>
> - Explicit VR Little Endian (ELE) — `1.2.840.10008.1.2.1` (Standard für verlustfreie Bildrahmen)
> - `transfer-syntax=*`Wenn ja, werden die Bildrahmen in der gespeicherten Übertragungssyntax zurückgegeben.
> - JPEG 2000 mit hohem Durchsatz und RPCL-Optionen Bildkomprimierung (nur verlustfrei) — `1.2.840.10008.1.2.4.202` — wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.202`
> - JPEG 2000 Lossless — `1.2.840.10008.1.2.4.90` — wenn die Instanz als Lossless gespeichert ist. HealthImaging

- JPEG-Baseline (Prozess 1): Standardübertragungssyntax für die verlustbehaftete 8-Bit-JPEG-Bildkomprimierung `1.2.840.10008.1.2.4.50` —, wenn die Instanz gespeichert ist in HealthImaging `1.2.840.10008.1.2.4.50`
- JPEG 2000-Bildkomprimierung — `1.2.840.10008.1.2.4.91` — wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.91`
- JPEG 2000-Bildkomprimierung mit hohem Durchsatz `1.2.840.10008.1.2.4.203` - - wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.203`
- JPEG XL-Bildkomprimierung `1.2.840.10008.1.2.4.112` - - wenn die Instanz gespeichert ist unter HealthImaging `1.2.840.10008.1.2.4.112`
- Instanzen, in HealthImaging denen ein oder mehrere Bildrahmen gespeichert sind, die in der MPEG-Familie von Transfersyntaxen (zu der auch MPEG-4 AVC/H.264 and HEVC/H .265 gehören) kodiert sind MPEG2, können mit der entsprechenden Transfer-Syntax-UID abgerufen werden. Dies ist beispielsweise der Fall, `1.2.840.10008.1.2.4.100` wenn die Instanz als Main Profile Main Level gespeichert ist. MPEG2
- Möglicherweise erhalten Sie einen `NotAcceptableException` 406-Wert, wenn die angeforderte Übertragungssyntax aufgrund der gespeicherten Übertragungssyntax nicht zurückgegeben werden kann oder wenn es spezielle Verarbeitungswarnungen für die Instanz gibt. Wenn dies der Fall ist, versuchen Sie erneut, den Anruf mit`transfer-syntax=*`.

Weitere Informationen erhalten Sie unter Unterstützte Übertragungssyntaxen und Bibliotheken zur Dekodierung von Image-Frames für AWS HealthImaging.

DICOM-Massendaten abrufen von HealthImaging

Verwenden Sie die `GetDICOMBulkdata` Aktion, um Binärdaten abzurufen, die von DICOM-Metadaten in einem HealthImaging Datenspeicher getrennt wurden. Beim Abrufen von Instanz- oder Serienmetadaten werden binäre Attribute, die größer als 1 MB sind, durch A `BulkDataURI` statt durch Inline-Werte dargestellt. Sie können die Binärdaten für jeden primären Bilddatensatz im HealthImaging Datenspeicher abrufen, indem Sie die in der `BulkDataURI` Metadatenantwort angegebenen Daten verwenden. Sie können Massendaten für nicht primäre Bilddatensätze abrufen, indem Sie die Bilddatensatz-ID als Abfrageparameter angeben.

Um DICOM-Massendaten abzurufen

Wenn Sie DICOM-Metadaten aus einer HealthImaging DICOMweb WADO-RS-Aktion abrufen, wie z. B. `GetDICOMInstanceMetadata` oder`GetDICOMSeriesMetadata`, werden große binäre Attribute wie folgt ersetzt: BulkData URIs

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

Gehen Sie wie folgt vor, um ein DICOM-Element mit der `GetDICOMBulkdata` Aktion abzurufen.

1. Konstruieren Sie eine URL für die Anfrage mit den Werten aus dem`BulkDataURI`, des Formulars:

   ```
   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. `GetDICOMBulkdata`Geben Sie Ihren Befehl als HTTP-GET-Anfrage mit dem AWS Signature Version 4-Signaturprotokoll aus. Das folgende Codebeispiel verwendet das `curl` Befehlszeilentool, um ein DICOM-Element aus einem primären Bildsatz abzurufen:

   ```
   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'
   ```

   Um ein DICOM-Datenelement aus einem nicht-primären Bilddatensatz abzurufen, geben Sie einen Parameter an: `ImageSetId`

```
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

Der `imageSetId` Parameter ist erforderlich, um Massendaten für nicht primäre Bilddatensätze abzurufen. Die DICOMBulkdata Aktion Abrufen gibt nur dann Bulkdaten für primäre Bilddatensätze zurück, wenn die`datastoreId`, `studyInstanceUIDseriesInstanceUID`, und angegeben `SOPInstanceUID` sind (ohne ein). `imagesetID`

DICOM-Daten durchsuchen in HealthImaging

AWS HealthImaging bietet Darstellungen von QIDO-RSDICOMWeb-APIs, um anhand der Patienten-ID nach Studien, Serien und Instanzen zu suchen und deren eindeutige Identifikatoren für die weitere Verwendung zu erhalten. HealthImagingDie QIDO-RS DICOMWeb-APIs bieten Flexibilität bei der Suche nach in älteren Anwendungen gespeicherten Daten HealthImaging und sorgen für Interoperabilität mit älteren Anwendungen.

Wichtig

HealthImagingDie DICOMWeb-APIs können verwendet werden, um Bilddatensatzinformationen mit zurückzugeben. QIDO-RS HealthImaging DICOMWeb-APIs verweisen nur auf primäre Bilddatensätze, sofern nicht anders angegeben. Verwenden Sie HealthImaging Cloud-native Aktionen oder den optionalen Imageset-Parameter von DICOMWeb-Aktionen, um Bilddatensätze abzurufen, die nicht primär sind. HealthImagingDie

DICOMWeb-APIs können verwendet werden, um Bilddatensatzinformationen mit Antworten zurückzugeben. DICOMweb-conformant
HealthImaging QIDO-RS DICOMWeb-Aktionen können maximal 10.000 Datensätze zurückgeben. Falls mehr als 10.000 Ressourcen vorhanden sind, können sie nicht über die Aktionen abgerufen werden, sondern können über QIDO-RS WADO-RS DICOMWeb-Aktionen oder Cloud-native Aktionen abgerufen werden.
Die in diesem Abschnitt aufgeführten APIs entsprechen dem DICOMWeb () QIDO-RS - Standard für webbasierte medizinische Bildgebung. Sie werden nicht über SDKs AWS CLI angeboten. AWS

DICOMWeb-Such-APIs für HealthImaging

In der folgenden Tabelle werden alle HealthImaging Darstellungen der QIDO-RS DICOMWeb-APIs beschrieben, in denen Daten gesucht werden können. HealthImaging

HealthImaging Repräsentationen von DICOMWeb-APIs QIDO-RS

Name	Description
`SearchDICOMStudies`	Suchen Sie nach DICOM-Studien, HealthImaging indem Sie Suchabfrageelemente mithilfe einer GET-Anfrage angeben. Die Ergebnisse der Studiensuche werden im JSON-Format zurückgegeben, sortiert nach letzter Aktualisierung und absteigendem Datum (vom letzten zum ältesten). Siehe Suchen Sie nach Studien.
`SearchDICOMSeries`	Suchen Sie nach DICOM-Serien in, HealthImaging indem Sie Suchabfrageelemente mithilfe einer GET-Anfrage angeben. Die Suchergebnisse für Serien werden im JSON-Format zurückgegeben, sortiert nach `Series Number (0020, 0011)` aufsteigender Reihenfolge (vom ältesten zum neuesten). Siehe Suchen Sie nach Serien.

Name	Description
`SearchDICOMInstances`	Suchen Sie nach DICOM-Instanzen, HealthImaging indem Sie Suchabfrageelemente mithilfe einer GET-Anforderung angeben. Die Ergebnisse der Instanzsuche werden im JSON-Format zurückgegeben, sortiert nach `Instance Number (0020, 0013)` aufsteigender Reihenfolge (vom ältesten zum neuesten). Siehe Suchen Sie nach Instanzen.

Unterstützte DICOMWeb-Abfragetypen für HealthImaging

HealthImaging unterstützt QIDO-RS hierarchische Ressourcenabfragen auf den Ebenen Study, Series und SOP Instance. Bei Verwendung der QIDO-RS hierarchischen Suche nach: HealthImaging

- Bei der Suche nach Studien wird eine Liste von Studien zurückgegeben
- Die Suche nach einer Reihe einer Studie erfordert eine bekannte Datenreihe `StudyInstanceUID` und gibt eine Liste von Reihen zurück
- Für die Suche in einer Liste von Instanzen ist ein bekanntes `StudyInstanceUID` und `SeriesInstanceUID`

In der folgenden Tabelle werden unterstützte QIDO-RS hierarchische Abfragetypen für die Suche nach Daten beschrieben HealthImaging.

HealthImaging unterstützte QIDO-RS Abfragetypen

Abfragetyp	Beispiel
Abfragen von Attributwerten	Suchen Sie in einer Studie nach allen Serien, womodality=CT . `.../studies/1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894866/series?00080060=CT`

Abfragetyp	Beispiel
	Durchsucht alle Studien, bei denen die Patienten-ID und das Studiendatum diesen Werten entsprechen. `…/studies?PatientID=1123581 3&StudyDate=20130509`
Abfragen nach Schlüsselwörtern	Durchsuchen Sie alle Serien mit dem `SeriesInstanceUID` Schlüsselwort. `.../studies/1.3.6.1.4.1.145 19.5.2.1.6279.6001.10137060 527657755614301389486 6/seri es?SeriesInstanceUID=1.3.6. 1.4.1.14519.5.2.1.6279.6001 .101370605276577556143013894868`
Tag-Abfragen	Suchen Sie mithilfe von im group/element Formular übergebenen Abfrageparametern nach Tags. {Gruppe} {Element} wie 0020000D
Bereichsabfragen	`...?Modality=CT&StudyDate=A ABBYYYY-BBCCYYYY`
Pagieren von Ergebnissen mit und `limit offset`	`.../studies?limit=1&offset= 0&00080020=20000101` Sie können die Parameter Limit und Offset verwenden, um Suchantworten zu paginiere n. Der Standardwert für Limit ist 1000. Den Maximalwert HealthImaging AWS-Endpunkte und Kontingente finden Sie unter. Maximaler Grenzwert = 1000, maximaler Offset = 9000

Abfragetyp	Beispiel

Abfragen mit Platzhaltern	Platzhalterabfragen bieten mehr Flexibilität bei der Suche mit „*“ und „?“. „*“ entspricht einer beliebigen Zeichenfolge (einschließlich eines Längenwerts von Null) und „?“ entspricht einem beliebigen einzelnen Zeichen.
	
	Sucht nach allen Studien in einem Datenspeicher, der „Nuclear“ StudyDescription enthält:
	
	`.../studies?StudyDescription=*Nuclear*`
	
	Suche nach allen Studien, die mit „Nuclear“ StudyDescription enden:
	
	`.../studies?StudyDescription=*Nuclear`
	
	Suche nach allen Studien, die mit „Nuclear“ StudyDescription beginnen:
	
	`.../studies?StudyDescription=Nuclear*`
	
	Suchen Sie nach allen Studien, in denen PatientID genau 3 beliebige Zeichen nach 200965981 enthält:
	
	`.../studies?PatientID=200965981???`

Abfragetyp	Beispiel
FuzzyMatching Abfragen	Aktivieren Sie den Fuzzy-Matching für Namens-DICOM-Attribute ((0010,0010), PatientNa me ReferringPhysicianName (0008,0090)), indem Sie den optionalen Abfrageparameter Fuzzymatching hinzufügen: `.../studies?fuzzymatching=t rue&PatientName="Thomas^Albert"` Bei dieser Abfrage wird für jeden Teil des Werts ein Präfixabgleich ohne Berücksichtigung der Groß- und Kleinschreibung durchgeführt. PatientName Sie gibt Ergebnisse mit PatientNa me Werten wie „Thomas“, „Albert“, „Thomas Albert“, „Thomas^Albert“ zurück, aber nicht mit „hom“ oder „ber“.

Abfragetyp	Beispiel

IncludeField Abfragen	Verwenden Sie den `includefield` Abfrageparameter, um zusätzliche DICOM-Attribute anzufordern, die über den Standarda ntwortsatz hinausgehen.
	
	Gibt bestimmte Attribute nach Tag zurück:
	
	`.../studies?PatientID=11235 813&includefield=00101081&i ncludefield=PatientWeight`
	
	Gibt alle verfügbaren Attribute zurück:
	
	`.../studies?PatientID=11235 813&includefield=all`
	
	Gibt die Unterattribute der Sequenz (SQ) in gepunkteter Notation zurück:
	
	`.../studies?PatientID=11235 813&includefield=00080096.0 0080100`
	
	Private Datenelemente zurückgeben:
	
	`.../instances?includefield= 00191001&00190010=Philips`

Verwendung IncludeField in QIDO-RS Abfragen

Mit dem `includefield` Abfrageparameter können Sie zusätzliche DICOM-Attribute anfordern, die über die Standardantwortmenge in HealthImaging QIDO-RS Abfragen hinausgehen. Sie können ihn `includefield` auf Studien-, Serien- und Instanzebene verwenden.

Syntax

Verwenden Sie das folgende GET-Anforderungsformat, um zusätzliche Felder in Ihre QIDO-RS Abfragen aufzunehmen:

```
GET .../studies?<query_params>&includefield=<tag_or_keyword>
GET .../studies/<StudyInstanceUID>/series?<query_params>&includefield=<tag_or_keyword>
GET .../studies/<StudyInstanceUID>/series/<SeriesInstanceUID>/instances?
<query_params>&includefield=<tag_or_keyword>
```

Sie können mehrere `includefield` Parameter in einer einzigen Anfrage angeben:

```
GET .../studies?
PatientID=11235813&includefield=00101081&includefield=00101030&includefield=00101010
```

Unterstützte Werte für includefield

In der folgenden Tabelle werden die unterstützten Werte für den `includefield` Parameter beschrieben.

Unterstützte Include-Feldwerte

Werttyp	Description	Beispiel
DICOM-Tag (8 Hex-Zeichen)	Fordert ein bestimmtes DICOM-Attribut anhand seines Tags im GGGGEEEE-Format an.	`includefield=00081030`
`all`	Fordert alle verfügbaren DICOM-Attribute für die Ressourcenebene an.	`includefield=all`
Punktierter SQ-Pfad	<parent_tag>Fordert ein bestimmtes Unterattribut innerhalb eines Sequence (SQ) -Attributs mithilfe der Punktnotation an:. <child_tag>.	`includefield=00080096.00080100`
Tag für private Datenelemente	Fordern Sie ein privates Tag an (Element mit einer ungeraden Gruppe). Erfordert den Parameter`privateCreatorElement` .	`includefield=00191001`

Werttyp	Description	Beispiel
Standard-DICOM-Attribute, einschließlich Bulkdata	Fordern Sie bestimmte einzelne oder mehrere Attribute nach Tag oder Schlüsselwort an.	`includefield=00102201`

Verhalten und Regeln

Die folgenden Regeln gelten für `includefield` Abfragen:

- Standardantwort — Ohne `includefield` Angabe gibt die QIDO-RS Antwort nur die Standardattribute zurück.
- includefield=all — Gibt alle verfügbaren Attribute auf der angeforderten Ebene zurück. Wenn mit anderen `includefield` Werten kombiniert `all` wird, hat Vorrang. `all`
- Maximale Anzahl von Tags — Eine Anfrage kann bis zu 50 `includefield` Parameter enthalten.
- Doppelte Tags — Doppelte `includefield` Werte werden dedupliziert und als eine einzige Anfrage behandelt.
- Ungültige oder fehlende Tags — Wenn ein angefordertes Tag in den DICOM-Daten nicht vorhanden oder ungültig ist, wird es stillschweigend in der Antwort weggelassen. Andere gültige `includefield` Attribute werden weiterhin zurückgegeben.

Sequenzattribute (SQ)

Verwenden Sie die Punktnotation, um verschachtelte Attribute innerhalb eines Sequence-Attributs (SQ) anzufordern:

```
includefield=<parent_SQ_tag>.<child_tag>
```

Um beispielsweise CodeValue (0008.0100) innerhalb ReferringPhysicianIdentificationSequence von (0008.0096) abzurufen:

```
GET .../studies?PatientID=11235813&includefield=00080096.00080100
```

Multi-level Verschachtelung wird unterstützt. Beispiel:

```
includefield=00081115.00081199.00081150
```

Private Stichwörter

Private DICOM-Datenelemente (ODD-Group-Tags) werden auf allen Ressourcenebenen unterstützt. Um private Tags anzufordern, geben Sie den Abfrageparameter an`privateCreatorElement`.

Verwenden Sie die folgende Syntax:

```
GET .../instances?includefield=<private_tag>&<creator_tag>=<creator_name>
```

Beispiel:

```
GET .../instances?includefield=00191001&00190010=Philips
```

Die folgenden Regeln gelten für private Tags:

- Das `privateCreatorElement` Tag und der Name des Erstellers müssen als Vergleichsparameter angegeben werden, wenn das private Tag angefordert wird.
- Wenn das angegebene Tag nicht gefunden `privateCreatorElement` wird, wird das private Tag stillschweigend weggelassen.
- Wenn nur ein `privateCreatorElement` Tag ohne ein privates Datenelement angefordert wird, werden nur der Name und der Wert des Erstellerelements zurückgegeben. Es werden nicht alle Tags zurückgegeben, die zum Block dieses Erstellers gehören.

Massendaten-Tags

DICOM-Attribute mit Bulkdata-VRs (OB, OD, OF, OL, UN, OW, OV), deren Binärwert größer als 1 MB ist, als Teil von angefordert wurden, `includefield` werden `bulkdataURI` anstelle des rohen Binärwerts als zurückgegeben. Weitere Informationen zum Abrufen von Massendaten finden Sie unter DICOM-Massendaten abrufen in. HealthImaging

Was gibt includefield=all auf jeder Ebene zurück?

Wenn `includefield=all` angegeben, umfasst die Antwort alle Attribute auf der jeweiligen Ressourcenebene.

Studienebene (includefield=all)

In der folgenden Tabelle sind alle Attribute aufgeführt, die auf der Studienebene zurückgegeben wurden, sofern diese angegeben wurde. `includefield=all`

Attribute auf Studienebene für includefield=all

Markierung	Name	VR
00080005	SpecificCharacterSet	CS
00080020	StudyDate	TAG
00080030	StudyTime	TM
00080050	AccessionNumber	SH
00080051	IssuerOfAccessionNumberSequence	Q
00080056	InstanceAvailability	CS
00080061	ModalitiesInStudy	CS
00080062	SOPClassesInStudy	Benutzeroberfläche
00080090	ReferringPhysicianName	PN
0008009C	ConsultingPhysicianName	PN
00080201	TimezoneOffsetFromUTC	SH
00081030	StudyDescription	VIEL
00081048	PhysiciansOfRecord	PN
00081060	NameOfPhysiciansReadingStudy	PN
00081080	AdmittingDiagnosesDescription	VIEL
00081190	URL abrufen	UR
00100010	PatientName	PN
00100020	ID des Patienten	LO
00100021	IssuerOfPatientID	NIEDRIG

Markierung	Name	VR
00100022	TypeOfPatientID	CS
00100026	SourcePatientGroupIdentificationSequence	Q
00100027	GroupOfPatientsIdentificationSequence	Q
00100028	SubjectRelativePositionInImage	US
00100030	PatientBirthDate	TAG
00100032	PatientBirthTime	TM
00100033	PatientBirthDateInAlternativeCalendar	NIEDRIG
00100034	PatientDeathDateInAlternativeCalendar	NIEDRIG
00100035	PatientAlternativeCalendar	CS
00100040	PatientSex	CS
00100050	PatientInsurancePlanCodeSequence	Q
00100101	PatientPrimaryLanguageCodeSequence	Q
00100102	PatientPrimaryLanguageModifierCodeSequence	Q
00100200	QualityControlSubject	CS
00100201	QualityControlSubjectTypeCodeSequence	Q
00100213	StrainNomenclature	VIEL
00100214	StrainStockNumber	VIEL
00100215	StrainSourceRegistryCodeSequence	Q
00100217	StrainSource	VIEL
00100219	StrainCodeSequence	Q

Markierung	Name	VR
00100223	GeneticModificationsNomenclature	NIEDRIG
00100229	GeneticModificationsCodeSequence	Q
00101001	OtherPatientNames	PN
00101005	PatientBirthName	PN
00101010	PatientAge	AS
00101020	PatientSize	DS
00101021	PatientSizeCodeSequence	Q
00101022	PatientBodyMassIndex	DS
00101023	Gemessene AP-Abmessung	DS
00101024	MeasuredLateralDimension	DS
00101030	PatientWeight	DS
00101040	PatientAddress	VIEL
00101060	PatientMotherBirthName	PN
00101080	MilitaryRank	VIEL
00101081	BranchOfService	VIEL
00102000	MedicalAlerts	VIEL
00102110	Allergien	LO
00102150	CountryOfResidence	VIEL
00102152	RegionOfResidence	VIEL
00102154	PatientTelephoneNumbers	SH

Markierung	Name	VR
00102160	EthnicGroup	SH
00102180	Beruf	SH
001021A0	SmokingStatus	CS
001021C0	PregnancyStatus	US
001021D0	LastMenstrualDate	TAG
001021F0	PatientReligiousPreference	VIEL
00102201	PatientSpeciesDescription	VIEL
00102202	PatientSpeciesCodeSequence	Q
00102203	PatientSexNeutered	CS
00102210	AnatomicalOrientationType	CS
00102292	PatientBreedDescription	VIEL
00102293	PatientBreedCodeSequence	Q
00102295	BreedRegistrationNumber	VIEL
00102296	BreedRegistryCodeSequence	Q
00102297	ResponsiblePerson	PN
00102298	ResponsiblePersonRole	CS
00102299	ResponsibleOrganization	NIEDRIG
00109431	ExaminedBodyThickness	FL
0020000D	StudyInstanceUID	Benutzeroberfläche
00200010	Studien-ID	SH

Markierung	Name	VR
00201206	NumberOfStudyRelatedSeries	IS
00201208	NumberOfStudyRelatedInstances	IS
00321032	RequestingPhysician	PN
00321033	RequestingService	NIEDRIG
00321060	RequestedProcedureDescription	VIEL
00321070	RequestedContrastAgent	VIEL
00380010	Zulassungsnummer	LO
00380016	RouteOfAdmissions	VIEL
00380020	AdmittingDate	TAG
00380021	AdmittingTime	TM
00380050	SpecialNeeds	NIEDRIG
00380060	ServiceEpisodeID	LO
00380062	ServiceEpisodeDescription	VIEL
00380300	CurrentPatientLocation	NIEDRIG
00380400	PatientInstitutionResidence	NIEDRIG
00380500	PatientState	LO
00400244	PerformedProcedureStepStartDate	TAG
00400245	PerformedProcedureStepStartTime	TM
00400250	PerformedProcedureStepEndDate	TAG
00400251	PerformedProcedureStepEndTime	TM

Markierung	Name	VR
00400253	PerformedProcedureStepID	SH
00081032	ProcedureCodeSequence	Q
00100024	IssuerOfPatientIDQualifiersSequence	Q
00321034	RequestingServiceCodeSequence	Q
00321064	RequestedProcedureCodeSequence	Q
00401012	ReasonForPerformedProcedureCodeSequence	Q

Serienebene (includefield=all)

In der folgenden Tabelle sind die Attribute auf Serienebene aufgeführt, die zurückgegeben werden, wenn sie angegeben ist. `includefield=all` Die Datenreihenebene gibt auch alle in der vorherigen Tabelle aufgeführten Attribute auf Studienebene zurück.

Attribute auf Reihenebene für includefield=all

Markierung	Name	VR
00080021	SeriesDate	DA
00080031	SeriesTime	TM
00080060	Modalität	CS
00080064	ConversionType	CS
00080068	PresentationIntentType	CS
00080070	Hersteller	LO
00080080	InstitutionName	LO
00080082	InstitutionCodeSequence	Q
00081010	StationName	SH

Markierung	Name	VR
0008103E	SeriesDescription	VIEL
0008103F	SeriesDescriptionCodeSequence	Q
00081040	InstitutionalDepartmentName	NIEDRIG
00081041	InstitutionalDepartmentTypeCodeSequence	Q
00081050	PerformingPhysicianName	PN
00081070	OperatorsName	PN
00081090	ManufacturerModelName	NIEDRIG
00180010	ContrastBolusAgent	VIEL
00180015	BodyPartExamined	CS
00180050	SliceThickness	DS
00180088	SpacingBetweenSlices	DS
00181000	DeviceSerialNumber	NIEDRIG
00181016	SecondaryCaptureDeviceManufacturer	VIEL
00181018	SecondaryCaptureDeviceManufacturerModelName	VIEL
00181019	SecondaryCaptureDeviceSoftwareVersions	VIEL
00181020	SoftwareVersions	VIEL
00181030	ProtocolName	VIEL
00181050	SpatialResolution	DS
00181200	DateOfLastCalibration	TAG
00181201	TimeOfLastCalibration	TM

Markierung	Name	VR
00185100	PatientPosition	CS
0020000D	StudyInstanceUID	Benutzeroberfläche
0020000E	SeriesInstanceUID	Benutzeroberfläche
00200011	SeriesNumber	IS
00200052	FrameOfReferenceUID	Benutzeroberfläche
00200060	Lateralität	CS
00201209	NumberOfSeriesRelatedInstances	IS
00540081	NumberOfSlices	US
00540101	NumberOfTimeSlices	US
00541000	SeriesType	CS

Instanzebene (includefield=all)

Gibt auf Instanzebene die vollständigen DICOM-Metadaten auf Instanzebene `includefield=all` zurück. Dies schließt alle Attribute ein, die in den Instanzmetadaten im Speicher gespeichert sind. HealthImaging Jedes DICOM-Tag, das in der ursprünglichen DICOM-Datei für diese Instanz vorhanden ist, wird zurückgegeben, mit Ausnahme des Pixeldatenattributs.

Themen

- Auf der Suche nach DICOM-Studien in HealthImaging
- Auf der Suche nach der DICOM-Serie in HealthImaging
- Suche nach DICOM-Instanzen in HealthImaging

Auf der Suche nach DICOM-Studien in HealthImaging

Verwenden Sie die `SearchDICOMStudies` API, um in einem Datenspeicher nach DICOM-Studien zu suchen. HealthImaging Sie können in nach DICOM-Studien suchen, HealthImaging indem Sie eine URL erstellen, die unterstützte DICOM-Datenelemente (Attribute) enthält. Die Ergebnisse der Studiensuche werden im JSON-Format zurückgegeben, sortiert nach letzter Aktualisierung und absteigendem Datum (vom letzten zum ältesten).

Um nach DICOM-Studien zu suchen

1. Sammeln HealthImaging `region` und bewerten`datastoreId`. Weitere Informationen finden Sie unter Eigenschaften des Datenspeichers abrufen.
2. Erstellen Sie eine URL für die Anfrage, einschließlich aller zutreffenden Studienelemente. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL-Pfad im folgenden Beispiel anzuzeigen. Die URL hat die Form:

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

   Studienelemente für **`SearchDICOMStudies`**

DICOM-Element-Tag	Name des DICOM-Elements
(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

DICOM-Element-Tag	Name des DICOM-Elements
(0010,0030)	Patient BirthDate
(0010,0032)	Patient BirthTime
(0020,000D)	Study Instance UID
(0020,0010)	Study ID

3. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `SearchDICOMStudies`verwendet eine HTTP-GET-Anfrage mit dem AWS Signaturprotokoll Signature Version 4. Im folgenden Beispiel wird das `curl` Befehlszeilentool verwendet, um nach Informationen über DICOM-Studien zu suchen.

   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
   ```

   Die Ergebnisse der Studiensuche werden im JSON-Format zurückgegeben, sortiert nach letzter Aktualisierung und absteigendem Datum (vom letzten zum ältesten).

Auf der Suche nach der DICOM-Serie in HealthImaging

Verwenden Sie die `SearchDICOMSeries` API, um in einem Datenspeicher nach der DICOM-Serie zu suchen. HealthImaging Sie können in nach DICOM-Serien suchen, HealthImaging indem Sie eine URL erstellen, die unterstützte DICOM-Datenelemente (Attribute) enthält. Die Suchergebnisse für Serien werden im JSON-Format in aufsteigender Reihenfolge (vom ältesten zum neuesten) zurückgegeben.

Um nach der DICOM-Serie zu suchen

1. Sammeln HealthImaging `region` und bewertendatastoreId. Weitere Informationen finden Sie unter Eigenschaften des Datenspeichers abrufen.
2. Sammle den `StudyInstanceUID` Wert. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen abrufen.
3. Konstruieren Sie eine URL für die Anfrage, einschließlich aller zutreffenden Series-Elemente. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL-Pfad im folgenden Beispiel anzuzeigen. Die URL hat die Form:

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

Serienelemente für **`SearchDICOMSeries`**

DICOM-Element-Tag	Name des DICOM-Elements
`(0008,0060)`	`Modality`
`(0020,000E)`	`Series Instance UID`

4. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `SearchDICOMSeries`verwendet eine HTTP-GET-Anfrage mit dem AWS Signaturprotokoll Signature Version 4. Im folgenden Beispiel wird das `curl` Befehlszeilentool verwendet, um nach Informationen der DICOM-Serie zu suchen.

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
```

Die Suchergebnisse für Serien werden im JSON-Format zurückgegeben, sortiert nach `Series Number (0020,0011)` aufsteigender Reihenfolge (vom ältesten zum neuesten).

Suche nach DICOM-Instanzen in HealthImaging

Verwenden Sie die `SearchDICOMInstances` API, um in einem Datenspeicher nach DICOM-Instanzen zu suchen. HealthImaging Sie können in nach DICOM-Instanzen suchen, HealthImaging indem Sie eine URL erstellen, die unterstützte DICOM-Datenelemente (Attribute) enthält. Die Instanzergebnisse werden im JSON-Format in aufsteigender Reihenfolge (vom ältesten zum neuesten) zurückgegeben.

Um nach DICOM-Instanzen zu suchen

1. Sammeln HealthImaging `region` und `datastoreId` Werte. Weitere Informationen finden Sie unter Eigenschaften des Datenspeichers abrufen.
2. Sammle Werte für `StudyInstanceUID` und`SeriesInstanceUID`. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen abrufen.
3. Konstruieren Sie eine URL für die Anfrage, einschließlich aller zutreffenden Suchelemente. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL-Pfad im folgenden Beispiel anzuzeigen. Die URL hat die Form:

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

   Instanzelemente für **`SearchDICOMInstances`**

DICOM-Element-Tag	Name des DICOM-Elements
`(0008,0016)`	`SOP Class UID`
`(0008,0018)`	`SOP Instance UID`
`(0008,1196)`	`WarningReason`

   HealthImaging verwendet das DICOM-Element (0008,1196), um Import-Warncodes dauerhaft zu speichern. Die Import-Warncodes können auf Instanzebene durchsucht werden. Importwarncodes können mit Platzhaltern oder bestimmten Warncodes durchsucht werden. Siehe HealthImaging Warncodes.
4. Bereiten Sie Ihre Anfrage vor und senden Sie sie ab. `SearchDICOMInstances`verwendet eine HTTP-GET-Anfrage mit dem AWS Signaturprotokoll Signature Version 4. Im folgenden Beispiel

wird das `curl` Befehlszeilentool verwendet, um nach Informationen über DICOM-Instanzen zu suchen.

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
```

Die Ergebnisse der Instanzsuche werden im JSON-Format zurückgegeben, sortiert nach `Instance Number (0020,0013)` aufsteigender Reihenfolge (vom ältesten zum neuesten)

OIDC-Authentifizierung für DICOMweb APIs

AWS HealthImaging unterstützt zusätzlich zur bestehenden AWS Signature Version 4-Authentifizierung (Sigv4) die OAuth 2.0-basierte Authentifizierung für DICOMweb API-Anfragen mithilfe von OpenID Connect (OIDC). OIDC ermöglicht Ihnen die HealthImaging direkte Integration mit externen Identitätsanbietern (IdPs) und ermöglicht es Ihnen, standardbasierten Anwendungen über HealthImaging DICOMweb Endgeräte Zugriff auf Ihre medizinischen Bildgebungsdaten zu gewähren, ohne dass jede Anwendung über Anmeldeinformationen verfügen muss. AWS

Themen

- Benutzerdefinierte Token-Verifizierung mit Lambda-Autorisierern
- Richten Sie einen AWS Lambda-Authorizer für die OIDC-Authentifizierung ein

Benutzerdefinierte Token-Verifizierung mit Lambda-Autorisierern

HealthImaging implementiert OIDC-Unterstützung über eine Architektur, die Lambda-Autorisierer verwendet, sodass Kunden ihre eigene Token-Verifizierungslogik implementieren können. Dieses Design bietet Ihnen eine flexible Kontrolle darüber, wie Token validiert und wie Zugriffsentscheidungen durchgesetzt werden, und trägt so einer vielfältigen Landschaft von

OIDC-kompatiblen Identitätsanbietern () und unterschiedlichen Methoden zur Token-Verifizierung Rechnung. IdPs

Authentifizierungsfluss

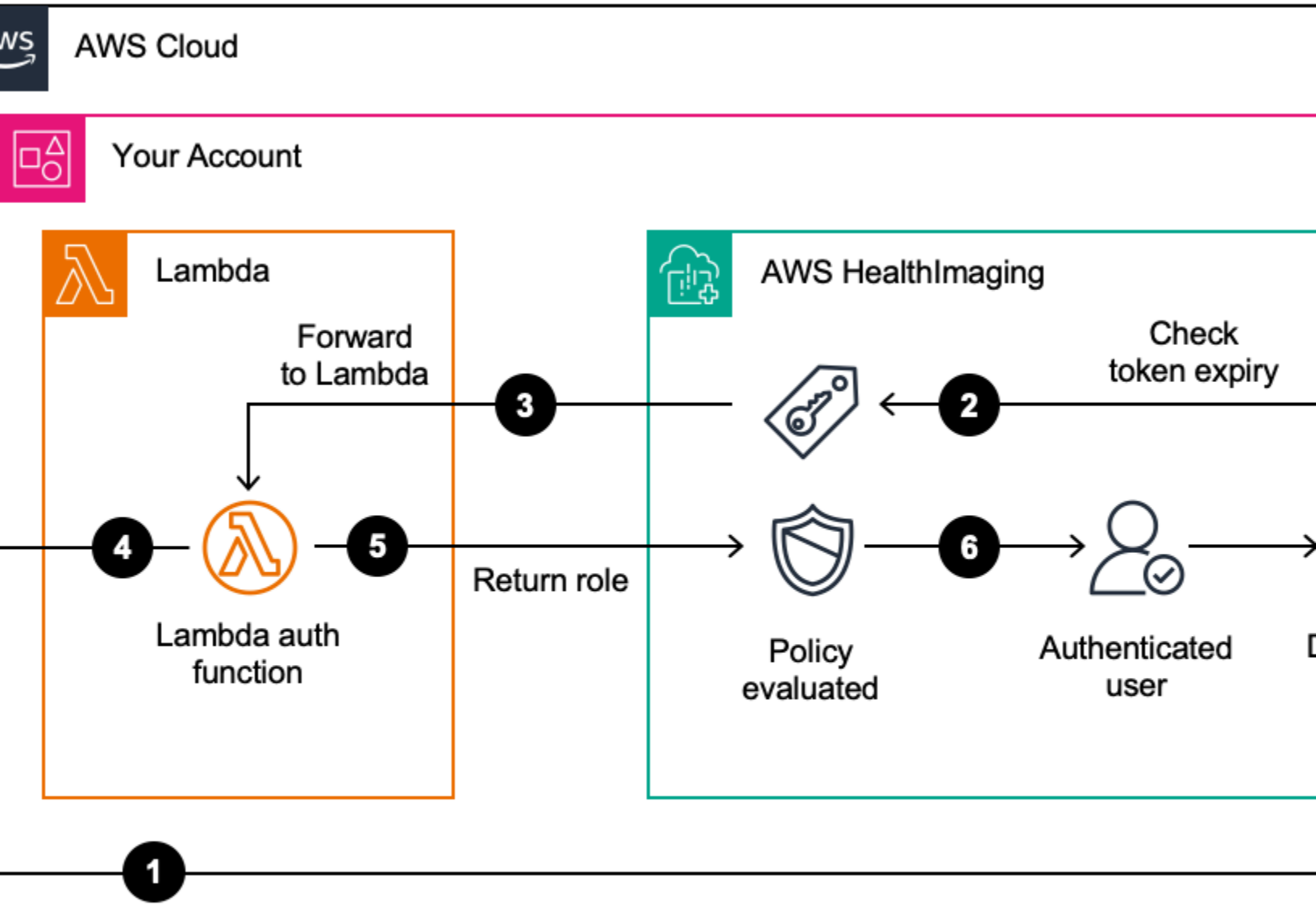
So funktioniert die Authentifizierung auf hoher Ebene:

1. Der Client ruft die DICOMweb API auf: Ihre Anwendung authentifiziert sich bei Ihrem ausgewählten OIDC-Identitätsanbieter und erhält ein signiertes ID-Token (JWT). Für jede DICOMweb HTTP-Anfrage muss der Client das OIDC-Zugriffstoken in den Authorization-Header aufnehmen (normalerweise ein Bearer-Token). Bevor die Anfrage Ihre Daten erreicht, HealthImaging extrahiert Sie dieses Token aus der eingehenden Anfrage und ruft einen Lambda-Autorisierer auf, den Sie konfigurieren.
   a. Der Header folgt normalerweise dem Format:. `Authorization: Bearer <token>`
2. Erste Überprüfung: HealthImaging Überprüft Ansprüche auf Zugriffstoken, um offensichtlich ungültige oder abgelaufene Token schnell zurückzuweisen, ohne die Lambda-Funktion unnötig aufzurufen. HealthImaging führt eine erste Überprüfung bestimmter Standardansprüche im Zugriffstoken durch, bevor der Lambda-Autorisierer aufgerufen wird:
   a. `iat`(Ausgestellt am): HealthImaging prüft, ob die Ausstellungszeit des Tokens innerhalb akzeptabler Grenzen liegt.
   b. `exp`(Ablaufzeit): HealthImaging Überprüft, ob das Token nicht abgelaufen ist.
   c. `nbf`(Nicht vor der Uhrzeit): Falls vorhanden, wird HealthImaging sichergestellt, dass das Token nicht vor seiner gültigen Startzeit verwendet wird.
3. HealthImaging ruft einen Lambda-Autorisierer auf: Wenn die anfängliche Überprüfung des Antrags erfolgreich ist, HealthImaging wird die weitere Token-Überprüfung an die vom Kunden konfigurierte Lambda-Autorisierungsfunktion delegiert. HealthImaging übergibt das extrahierte Token und andere relevante Anforderungsinformationen an die Lambda-Funktion. Die Lambda-Funktion überprüft die Signatur und die Ansprüche des Tokens.
4. Verifizierung mit einem Identitätsanbieter: Das Lambda enthält benutzerdefinierten Code, der die ID-Token-Signatur überprüft, eine umfassendere Tokenverifizierung durchführt (z. B. Aussteller, Zielgruppe, benutzerdefinierte Ansprüche) und diese Ansprüche bei Bedarf gegenüber dem IdP validiert.
5. Der Authorizer gibt eine Zugriffsrichtlinie zurück: Nach erfolgreicher Überprüfung bestimmt die Lambda-Funktion die entsprechenden Berechtigungen für die authentifizierte Verwendung. Der Lambda-Autorisierer gibt dann den Amazon-Ressourcennamen (ARN) einer IAM-Rolle zurück, der den Satz der zu erteilenden Berechtigungen darstellt.

6. Ausführung der Anfrage: Wenn die angenommene IAM-Rolle über die erforderlichen Berechtigungen verfügt, wird mit der Rückgabe der HealthImaging angeforderten Ressource fortgefahren. DICOMWeb Wenn die Berechtigungen nicht ausreichen, HealthImaging lehnt die Anfrage ab und gibt eine entsprechende Fehlermeldung zurück (d. h. 403 Forbidden).

> **Note**
>
> Die Autorisierungs-Lambda-Funktion wird nicht vom HealthImaging AWS-Service verwaltet. Sie wird in Ihrem Konto ausgeführt. AWS Den Kunden werden der Funktionsaufruf und die Ausführungszeit separat von ihren HealthImaging Gebühren in Rechnung gestellt.

Übersicht über die Architektur



OIDC-Authentifizierungsworkflow mit Lambda Authorizer

Voraussetzungen

Anforderungen an Zugriffstoken

HealthImaging erfordert, dass das Zugriffstoken im Format JSON Web Token (JWT) vorliegt. Viele Identity Provider (IDPs) bieten dieses Token-Format nativ an, während Sie bei anderen das Zugriffstoken-Formular auswählen oder konfigurieren können. Stellen Sie sicher, dass der von Ihnen gewählte IDP JWT-Token ausstellen kann, bevor Sie mit der Integration fortfahren.

Token-Format

Das Zugriffstoken muss im JWT-Format (JSON Web Token) vorliegen

Erforderliche Ansprüche

`exp`(Ablaufzeit)

Erforderlicher Anspruch, der angibt, wann das Token ungültig wird.

- Muss nach der aktuellen Uhrzeit in UTC liegen
- Stellt dar, wann das Token ungültig wird

`iat`(Ausgestellt am)

Erforderlicher Anspruch, der angibt, wann das Token ausgestellt wurde.

- Muss vor der aktuellen Uhrzeit in UTC liegen
- Darf NICHT früher als 12 Stunden vor der aktuellen Uhrzeit in UTC liegen
- Dadurch wird effektiv eine maximale Token-Lebensdauer von 12 Stunden durchgesetzt

`nbf`(Nicht vor der Zeit)

Optionaler Anspruch, der angibt, wann das Token frühestens verwendet werden kann.

- Falls vorhanden, wird es bewertet von HealthImaging
- Gibt den Zeitpunkt an, vor dem das Token nicht akzeptiert werden darf

Anforderungen an die Reaktionszeit des Lambda-Autorisierers

HealthImaging setzt strenge Zeitanforderungen für die Antworten des Lambda-Autorisierers durch, um eine optimale API-Leistung sicherzustellen. Ihre Lambda-Funktion muss innerhalb von 1 Sekunde zurückkehren.

Best Practices

Optimieren Sie die Token-Verifizierung

- Zwischenspeichern Sie JWKS (JSON Web Key Sets), wenn möglich
- Wenn möglich, gültige Zugriffstoken zwischenspeichern
- Minimiere die Anzahl der Netzwerkanrufe an deinen Identity Provider
- Implementieren Sie eine effiziente Token-Validierungslogik

Lambda-Konfiguration

- Auf Python und Node.js basierende Funktionen werden normalerweise schneller initialisiert
- Reduzieren Sie die Anzahl der zu ladenden externen Bibliotheken
- Konfigurieren Sie die entsprechende Speicherzuweisung, um eine konsistente Leistung zu gewährleisten
- Überwachen Sie die Ausführungszeiten mithilfe von CloudWatch Metriken

Aktivierung der OIDC-Authentifizierung

- Die OIDC-Authentifizierung kann nur aktiviert werden, wenn ein neuer Datenspeicher erstellt wird
- Die Aktivierung von OIDC für bestehende Datenspeicher wird über die API nicht unterstützt
- Um OIDC auf einem vorhandenen Datenspeicher zu aktivieren, müssen sich Kunden an den Support wenden AWS

Richten Sie einen AWS Lambda-Authorizer für die OIDC-Authentifizierung ein

In diesem Handbuch wird davon ausgegangen, dass Sie den Identity Provider (IdP) Ihrer Wahl bereits so konfiguriert haben, dass er Zugriffstoken bereitstellt, die mit den Anforderungen der HealthImaging OIDC-Authentifizierungsfunktion kompatibel sind.

1. Konfigurieren Sie IAM-Rollen für den API-Zugriff DICOMWeb

Erstellen Sie vor der Konfiguration des Lambda-Autorisierers IAM-Rollen, die bei der Verarbeitung DICOMWeb von HealthImaging API-Anfragen verwendet werden sollen. Die Autorisierungs-Lambda-

Funktion gibt nach erfolgreicher Token-Überprüfung den ARN einer dieser Rollen zurück, sodass HealthImaging die Anfragen mit den entsprechenden Berechtigungen ausgeführt werden können.

1. Erstellen Sie IAM-Richtlinien, die die gewünschten DICOMWeb API-Rechte definieren. Die verfügbaren Berechtigungen finden Sie im Abschnitt DICOMweb „Verwenden“ der HealthImaging Dokumentation.
2. Erstellen Sie IAM-Rollen, die:
   - Hängen Sie diese Richtlinien an
   - Schließen Sie eine Vertrauensbeziehung ein, die es dem HealthImaging AWS-Service Principal (`medical-imaging.amazonaws.com`) ermöglicht, diese Rollen zu übernehmen.

Hier ist ein Beispiel für eine Richtlinie, die zugehörigen Rollen den Zugriff auf eine HealthImaging DICOMWeb schreibgeschützte API ermöglicht:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "MedicalImagingDicomWebOperations",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMInstances",
                "medical-imaging:GetImageSetMetadata",
                "medical-imaging:GetDICOMSeriesMetadata",
                "medical-imaging:SearchDICOMStudies",
                "medical-imaging:GetDICOMBulkdata",
                "medical-imaging:SearchDICOMSeries",
                "medical-imaging:GetDICOMInstanceMetadata",
                "medical-imaging:GetDICOMInstance",
                "medical-imaging:GetDICOMInstanceFrames"
            ],
            "Resource": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/datastore-123"
        }
    ]
}
```

Hier ist ein Beispiel für die Vertrauensstellungsrichtlinie, die den Rollen zugeordnet werden sollte:

JSON

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

Der Lambda-Autorisierer, den Sie im nächsten Schritt erstellen, kann die Token-Ansprüche auswerten und den ARN der entsprechenden Rolle zurückgeben. AWS HealthImaging gibt sich dann als diese Rolle aus, um die DICOMWeb API-Anfrage mit den entsprechenden Berechtigungen auszuführen.

Beispiel:

- Ein Token mit „admin“ -Ansprüchen kann einen ARN für eine Rolle mit Vollzugriff zurückgeben
- Ein Token mit „Reader“ -Ansprüchen kann einen ARN für eine Rolle mit schreibgeschütztem Zugriff zurückgeben
- Ein Token mit den Ansprüchen „Department_A“ kann einen ARN für eine Rolle zurückgeben, die für die Zugriffsebene dieser Abteilung spezifisch ist

Dieser Mechanismus ermöglicht es Ihnen, das Autorisierungsmodell Ihres IdP über IAM-Rollen bestimmten HealthImaging AWS-Berechtigungen zuzuordnen.

2. Lambda Authorizer-Funktion erstellen und konfigurieren

Erstellen Sie eine Lambda-Funktion, die das JWT-Token verifiziert und den entsprechenden IAM-Rollen-ARN auf der Grundlage der Bewertung der Token-Ansprüche zurückgibt. Diese Funktion

wird vom Health Imaging Service aufgerufen und hat ein Ereignis übergeben, das die HealthImaging Datenspeicher-ID, den DICOMWeb Vorgang und das Zugriffstoken enthält, die in der HTTP-Anfrage gefunden wurden:

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

Die Lambda-Autorisierungsfunktion muss eine JSON-Antwort mit der folgenden Struktur zurückgeben:

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

Weitere Informationen finden Sie im Implementierungsbeispiel.

> **Note**
>
> Da die DICOMWeb Anfrage erst beantwortet wird, nachdem das Zugriffstoken vom Lambda-Autorisierer verifiziert wurde, ist es wichtig, dass die Ausführung dieser Funktion so schnell wie möglich erfolgt, um die beste DICOMWeb API-Antwortzeit zu gewährleisten.

Damit der HealthImaging Dienst autorisiert werden kann, die Lambda-Autorisierungsfunktion aufzurufen, muss er über eine Ressourcenrichtlinie verfügen, die es dem Dienst ermöglicht HealthImaging , ihn aufzurufen. Diese Ressourcenrichtlinie kann im Berechtigungsmenü der Registerkarte „Lambda-Konfiguration“ oder mithilfe von: erstellt werden AWS CLI

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

Diese Ressourcenrichtlinie ermöglicht es dem HealthImaging Dienst, Ihren Lambda-Authorizer aufzurufen, wenn API-Anfragen authentifiziert DICOMWeb werden.

> **Note**
>
> Die Lambda-Ressourcenrichtlinie kann später mit einer "ArnLike" -Bedingung aktualisiert werden, die dem ARN eines bestimmten HealthImaging Datenspeichers entspricht.

Hier ist ein Beispiel für eine Lambda-Ressourcenrichtlinie:

JSON

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "LambaAuthorizer-HealthImagingInvokePermission",
      "Effect": "Allow",
      "Principal": {
        "Service": "medical-imaging.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:123456789012::function:
{LambdaAuthorizerFunctionName}",
      "Condition": {
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/datastore-123"
        }
      }
    }
  ]
}
```

3. Erstellen Sie einen neuen Datenspeicher mit OIDC-Authentifizierung

Um die OIDC-Authentifizierung zu aktivieren, müssen Sie einen neuen Datenspeicher mit dem Parameter "" erstellen. AWS CLI lambda-authorizer-arn Die OIDC-Authentifizierung kann nicht für bestehende Datenspeicher aktiviert werden, ohne den Support zu kontaktieren. AWS

Hier ist ein Beispiel für die Erstellung eines neuen Datenspeichers mit aktivierter OIDC-Authentifizierung:

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

Sie können überprüfen, ob für einen bestimmten Datenspeicher die OIDC-Authentifizierungsfunktion aktiviert ist, indem Sie den Befehl AWS CLI get-datastore verwenden und überprüfen, ob das Attribut "" vorhanden ist: lambdaAuthorizerArn

```
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**
>
> Die Ausführungsrolle für den Befehl zur Erstellung des AWS CLI Datenspeichers muss über die entsprechenden Berechtigungen verfügen, um die Lambda-Autorisierungsfunktion aufzurufen. Dadurch werden Angriffe mit Rechteeskalation verhindert, bei denen böswillige Benutzer unbefugte Lambda-Funktionen über die Datastore-Authorizer-Konfiguration ausführen könnten.

Ausnahmecodes

Im Falle eines Authentifizierungsfehlers werden die folgenden HTTP-Fehlerantwortcodes und Textnachrichten HealthImaging zurückgegeben:

Bedingung	AHI-Antwort
Lambda Authorizer existiert nicht oder ist ungültig	424: Fehlkonfiguration des Authorizers
Der Authorizer wurde aufgrund eines Ausführungsfehlers beendet	424 Authorizer ist fehlgeschlagen
Irgendein anderer Authorizer-Fehler, der nicht zugeordnet wurde	424 Authorizer ist fehlgeschlagen
Der Autorisierer hat eine ungültige/falsch formulierte Antwort zurückgegeben	424 Fehlkonfiguration des Authorizers
Der Authorizer lief mehr als 1 Sekunde	408 Authorizer-Timeout
Das Token ist abgelaufen oder anderweitig ungültig	403 Ungültiges oder abgelaufenes Token
AHI kann die zurückgeg ebene IAM-Rolle aufgrund einer Fehlkonfiguration des Autorisierers nicht zusammenf ühren	4.2.4 Fehlkonfiguration des Autorisierers
Der Authorizer hat eine leere Rolle zurückgegeben	403 Zugriff verweigert
Die zurückgegebene Rolle ist nicht aufrufbar (Fehlkonf iguration Assume-Rolle/Trust)	424 Fehlkonfiguration des Autorisierers
Die Anforderungsrate überschreitet die Gateway-G renzen DICOMweb	429 Zu viele Anfragen

Bedingung	AHI-Antwort
Regionsübergreifender Datenspeicher, Rückgabef unktion oder Autorisierer Account/Cross	4.2.4 Regionsübergreifender Zugriff durch den Autorisierer Account/Cross

Beispiel für eine Implementierung

Dieses Python-Beispiel demonstriert eine Lambda-Autorisierungsfunktion, die AWS Cognito-Zugriffstoken anhand von HealthImaging Ereignissen überprüft und einen IAM-Rollen-ARN mit entsprechenden Rechten zurückgibt. DICOMWeb

Der Lambda-Authorizer implementiert zwei Caching-Mechanismen, um externe Aufrufe und Antwortlatenz zu reduzieren. Das JWKS (JSON Web Key Set) wird einmal pro Stunde abgerufen und im temporären Ordner der Funktion gespeichert, sodass es bei nachfolgenden Funktionsaufrufen lokal gelesen werden kann, anstatt es aus dem öffentlichen Netzwerk abzurufen. Sie werden auch feststellen, dass ein token_cache-Wörterbuchobjekt im globalen Kontext dieser Lambda-Funktion instanziiert wird. Globale Variablen werden von allen Aufrufen gemeinsam genutzt, die denselben erwärmten Lambda-Kontext wiederverwenden. Dadurch können erfolgreich verifizierte Token in diesem Wörterbuch gespeichert und bei der nächsten Ausführung derselben Lambda-Funktion schnell nachgeschlagen werden. Die Caching-Methode stellt einen generalistischen Ansatz dar, der für Zugriffstoken geeignet ist, die von den meisten Identitätsanbietern ausgegeben wurden. Eine AWS Cognito-spezifische Caching-Option finden Sie in den Abschnitten Benutzerpool verwalten und Caching in der AWS Cognito-Dokumentation.

```
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': ''
        }
```

Imaging-Daten mit AWS importieren HealthImaging

Beim Importieren werden Ihre medizinischen Bilddaten von einem Amazon S3 S3-Eingabe-Bucket in einen HealthImaging AWS-Datenspeicher verschoben. Während des Imports HealthImaging führt AWS eine Überprüfung der Pixeldaten durch, bevor Ihre DICOM P10-Dateien in Bildsätze umgewandelt werden, die aus Metadaten und Bildrahmen (Pixeldaten) bestehen.

> **Wichtig**
>
> HealthImaging Importaufträge verarbeiten DICOM-Instance-Binärdateien (`.dcm`Dateien) und wandeln sie in Bilddatensätze um. Verwenden Sie HealthImaging Cloud-native Aktionen (APIs), um Datenspeicher und Bilddatensätze zu verwalten. Verwenden Sie HealthImaging die Darstellung von DICOMweb Diensten, um DICOMweb Antworten zurückzugeben.

In den folgenden Themen wird beschrieben, wie Sie Ihre medizinischen Bilddaten mithilfe von AWS-Managementkonsole AWS CLI, und in einen HealthImaging Datenspeicher importieren AWS SDKs.

Themen

- Importaufträge verstehen
- Einen Importjob starten
- Eigenschaften von Importaufträgen abrufen
- Importaufträge auflisten

Importaufträge verstehen

Nachdem Sie einen Datenspeicher in AWS erstellt haben HealthImaging, müssen Sie Ihre medizinischen Bildgebungsdaten aus Ihrem Amazon S3 S3-Eingabe-Bucket in Ihren Datenspeicher importieren, um Bilddatensätze zu erstellen. Sie können die AWS SDKs AWS-Managementkonsole AWS CLI, und verwenden, um Importaufträge zu starten, zu beschreiben und aufzulisten.

Wenn Sie Ihre DICOM P10-Daten in einen HealthImaging AWS-Datenspeicher importieren, versucht der Service, Instanzen anhand der Metadatenelemente automatisch entsprechend der DICOM-Hierarchie von Study UID, Series UID und Instance UID zu organisieren. Importierte Daten werden als primär eingestuft, wenn die Metadatenelemente der importierten Daten nicht mit vorhandenen

primären Bilddatensätzen im Datenspeicher in Konflikt stehen. Wenn die Metadatenelemente neu importierter DICOM P10-Daten mit vorhandenen primären Bilddatensätzen in Konflikt stehen, werden die neuen Daten zu nicht primären Bilddatensätzen hinzugefügt. Wenn Datenimporte nicht primäre Bilddatensätze erzeugen, gibt HealthImaging AWS ein EventBridge Ereignis mit aus`isPrimary: False`, und der Datensatz, in den geschrieben wird, `success.ndjson` wird auch `isPrimary: False` im `importResponse` Objekt enthalten sein.

Wenn Sie Daten importieren, HealthImaging geht Folgendes vor:

- Wenn Instanzen, aus denen eine DICOM-Serie besteht, in einem Importauftrag importiert werden und die Instanzen nicht mit Instanzen kollidieren, die sich bereits im Datenspeicher befinden, dann sind alle Instanzen in einem primären Bildsatz organisiert.
- Wenn die Instanzen, aus denen eine DICOM-Serie besteht, in zwei oder mehr Importjobs importiert werden und die Instanzen nicht mit Instanzen kollidieren, die sich bereits im Datenspeicher befinden, dann sind alle Instanzen als ein primärer Imagesatz organisiert.
- Wenn eine Instanz mehr als einmal importiert wird, überschreibt die neueste Version alle älteren Versionen, die in einem primären Image-Set gespeichert sind, und die Versionsnummer des primären Image-Sets wird inkrementiert.

Sie können die Instanzen in der Primärversion mit den unter Aktualisieren von Image-Set-Metadaten beschriebenen Schritten aktualisieren.

Während des Imports werden Binärwerte in privaten Tags (mit den VR-Typen OB, OD, OF, OL, OV, OW, UN), die eine Größe von 1 MB überschreiten, getrennt von den Metadaten gespeichert. Beim Abrufen von Metadaten für diese Instanzen mithilfe von `GetDICOMInstanceMetadata` oder `GetDICOMSeriesMetadata` werden diese großen Binärwerte durch ersetzt BulkDataURIs, und die eigentlichen Binärdaten können mithilfe der `GetDICOMBulkdata` API abgerufen werden.

HealthImaging versucht, all Ihre medizinischen Bilddaten zu importieren. Wenn bei Importen Datenabweichungen oder unbekannte Datenelemente auftreten, HealthImaging fügt der `warning.ndjson` Datei Warnungen für DICOM-Instanzen hinzu, die trotzdem importiert werden könnten. Eine vollständige Liste der Warncodes finden Sie unter. HealthImaging Warncodes

Beachten Sie die folgenden Punkte, wenn Sie Ihre medizinischen Bildgebungsdateien aus Amazon S3 in einen HealthImaging Datenspeicher importieren:

- Die Instances, die einer DICOM-Serie entsprechen, werden automatisch in einem einzigen Bildsatz zusammengefasst, der als primär bezeichnet wird.

- Sie können DICOM P10-Daten in einem Importauftrag oder in mehreren Importaufträgen importieren, und der Service organisiert die Instanzen in primären Bilddatensätzen, die der DICOM-Serie entsprechen
- Beim Import gelten Längenbeschränkungen für bestimmte DICOM-Elemente. Um einen erfolgreichen Importvorgang sicherzustellen, stellen Sie sicher, dass Ihre medizinischen Bilddaten die Längenbeschränkungen nicht überschreiten. Weitere Informationen finden Sie unter Einschränkungen für DICOM-Elemente.
- Zu Beginn der Importaufträge wird eine Überprüfung der Pixeldaten durchgeführt. Weitere Informationen finden Sie unter Überprüfung der Pixeldaten.
- Importaktionen sind mit Endpunkten, Kontingenten und Drosselungslimits verknüpft HealthImaging . Weitere Informationen erhalten Sie unter Endpunkte und Kontingente und Drosselungsgrenzen.
- Für jeden Importauftrag werden die Verarbeitungsergebnisse am Standort gespeichert. `outputS3Uri` Die Verarbeitungsergebnisse sind als `job-output-manifest.json` Datei `SUCCESS` und `FAILURE` Ordner organisiert.

> **Note**
>
> Sie können bis zu 10.000 verschachtelte Ordner für einen einzelnen Importauftrag hinzufügen.

- Die `job-output-manifest.json` Datei enthält `jobSummary` Ausgaben und zusätzliche Details zu den verarbeiteten Daten. Das folgende Beispiel zeigt die Ausgabe aus einer `job-output-manifest.json` Datei.

```
{
"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": "123456123456123456789017891789012",
                "numberOfMatchedSOPInstances": 1
            }
        ]
    }
}
```

- Der SUCCESS Ordner enthält die `success.ndjson` Datei mit den Ergebnissen aller erfolgreich importierten Imaging-Dateien. Das folgende Beispiel zeigt die Ausgabe einer `success.ndjson` Datei.

```
{"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":"123456123456123456789017891789012", "isPrimary": True}}
```

- Der FAILURE Ordner enthält die `failure.ndjson` Datei mit den Ergebnissen aller Imaging-Dateien, die nicht erfolgreich importiert wurden. Das folgende Beispiel zeigt die Ausgabe einer `failure.ndjson` Datei.

```
{"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"}}
```

- Der `WARNING` Ordner enthält die `warning.ndjson` Datei mit den Ergebnissen aller Imaging-Dateien, die erfolgreich, aber mit Warnungen importiert wurden. Das folgende Beispiel zeigt die Ausgabe einer `warning.ndjson` Datei.

```
{"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."}]}}
```

- Importaufträge werden 90 Tage lang in der Auftragsliste aufbewahrt und anschließend archiviert.

Einen Importjob starten

Verwenden Sie die `StartDICOMImportJob` Aktion, um eine Überprüfung der Pixeldaten und den Massenimport von Daten in einen HealthImaging AWS-Datenspeicher zu starten. Der Importjob importiert DICOM P10-Dateien oder erweitert bestehende DICOM-Dateien mit JSON-Metadaten. Der `inputS3Uri` Parameter gibt den Amazon S3 S3-Eingabe-Bucket an, der die Quelldateien enthält. Die Ergebnisse der Importauftragsverarbeitung werden in dem durch den `outputS3Uri` Parameter angegebenen Amazon S3 S3-Ausgabe-Bucket gespeichert.

> **Note**
>
> Beachten Sie die folgenden Punkte, bevor Sie einen Importauftrag starten:
>
> - HealthImaging unterstützt den Import von DICOM P10-Dateien mit unterschiedlichen Übertragungssyntaxen. Einige Dateien behalten beim Import ihre ursprüngliche Übertragungssyntaxkodierung bei, während andere je nach Datenspeicherkonfiguration standardmäßig in HTJ2K Lossless oder JPEG 2000 Lossless transkodiert werden. Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen.
> - HealthImaging unterstützt Datenimporte aus Amazon S3 S3-Buckets, die sich in anderen unterstützten Regionen befinden. Um diese Funktionalität zu nutzen, geben Sie den `inputOwnerAccountId` Parameter an, wenn Sie einen Importjob starten. Weitere Informationen finden Sie unter Kontoübergreifender Import für AWS HealthImaging.
> - HealthImaging wendet beim Import Längenbeschränkungen auf bestimmte DICOM-Elemente an. Weitere Informationen finden Sie unter Einschränkungen für DICOM-Elemente.

- Um DICOM-Dateien mit JSON-Metadatenüberschreibungen zu importieren, geben Sie dem `importConfiguration` Parameter eine, `DicomMetadataMapping` die DICOM-Dateien ihren entsprechenden JSON-Metadatendateien zuordnet. Weitere Informationen finden Sie `StartDICOMImportJob`in der HealthImaging AWS-API-Referenz.

Die folgenden Menüs enthalten eine Vorgehensweise für die AWS SDKs AWS-Managementkonsole AWS CLI und Codebeispiele. Weitere Informationen finden Sie `StartDICOMImportJob`in der HealthImaging AWS-API-Referenz.

So starten Sie einen Importauftrag

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.
3. Wählen Sie DICOM-Daten importieren.

   Die Seite „DICOM-Daten importieren“ wird geöffnet.
4. Geben Sie im Abschnitt Details die folgenden Informationen ein:
   - Name (optional)
   - Quellspeicherort in S3 importieren
   - Konto-ID des Quell-Bucket-Besitzers (optional)
   - Verschlüsselungsschlüssel (optional)
   - Ausgabeziel in S3
5. Wählen Sie im Abschnitt Dienstzugriff die Option Bestehende Servicerolle verwenden und wählen Sie die Rolle aus dem Menü Servicerollenname aus, oder wählen Sie Neue Servicerolle erstellen und verwenden aus.
6. Wählen Sie Importieren aus.

AWS CLI und SDKs

C++

SDK für 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();
}
```

- Einzelheiten zur API finden Sie StartDICOMImportJobin der AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So starten Sie einen DICOM-Importauftrag

Im folgenden Codebeispiel für `start-dicom-import-job` wird ein DICOM-Importauftrage gestartet.

```
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"
```

Ausgabe:

```
{
```

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

Weitere Informationen finden Sie unter Starten eines Importauftrags im AWS HealthImaging Entwicklerhandbuch.

- Einzelheiten zur API finden Sie StartDICOMImportJobin der AWS CLI Befehlsreferenz.

Java

SDK für 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 "";
   }
```

- Einzelheiten zur API finden Sie StartDICOMImportJobin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input
 files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files
 are stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/",
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
```

```
    }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie StartDICOMImportJobin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie StartDICOMImportJobin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " 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.
```

- Einzelheiten zur API finden Sie StartDICOMImportJobin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Eigenschaften von Importaufträgen abrufen

Verwenden Sie die `GetDICOMImportJob` Aktion, um mehr über die Eigenschaften von HealthImaging AWS-Importaufträgen zu erfahren. Beispielsweise können Sie nach dem Start eines Importauftrags ausführen, `GetDICOMImportJob` um den Status des Auftrags zu ermitteln. Sobald der `jobStatus` zurückkehrt als`COMPLETED`, können Sie auf Ihre Bilddatensätze zugreifen.

> **Note**
>
> Das `jobStatus` bezieht sich auf die Ausführung des Importjobs. Daher kann ein Importauftrag auch dann ein `jobStatus` AS zurückgeben, `COMPLETED` wenn während des Importvorgangs Validierungsprobleme festgestellt werden. Wenn a als `jobStatus` zurückgegeben wird`COMPLETED`, empfehlen wir Ihnen dennoch, die in Amazon S3 geschriebenen Ausgabemanifeste zu überprüfen, da sie Details zum Erfolg oder Misserfolg einzelner P10-Objektimporte enthalten.

Die folgenden Menüs enthalten eine Vorgehensweise für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `GetDICOMImportJob`in der HealthImaging AWS-API-Referenz.

Um die Eigenschaften von Importaufträgen abzurufen

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Details zum Datenspeicher wird geöffnet. Die Registerkarte Bildsätze ist standardmäßig ausgewählt.
3. Wählen Sie die Registerkarte Importe.
4. Wählen Sie einen Importauftrag aus.

   Die Seite mit den Importauftragsdetails wird geöffnet und zeigt Eigenschaften des Importjobs an.

AWS CLI und SDKs

C++

SDK für 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;
}
```

- Einzelheiten zur API finden Sie unter Get DICOMImport Job in der AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So rufen Sie die Eigenschaften eines DICOM-Importauftrags ab

Im folgenden Codebeispiel für `get-dicom-import-job` werden die Eigenschaften eines DICOM-Importauftrags abgerufen.

```
aws medical-imaging get-dicom-import-job \
    --datastore-id "12345678901234567890123456789012" \
    --job-id "09876543210987654321098765432109"
```

Ausgabe:

```
{
    "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/1234567890123456789012345678901 2-
DicomImport-09876543210987654321098765432109/"
    }
}
```

Weitere Informationen finden Sie im AWS HealthImaging Developer Guide unter Abrufen der Eigenschaften von Importaufträgen.

- API-Einzelheiten finden Sie unter Get DICOMImport Job in der AWS CLI Befehlsreferenz.

Java

SDK für 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;
   }
```

- Einzelheiten zur API finden Sie unter Get DICOMImport Job in der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
```

```
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie unter Get DICOMImport Job in der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- API-Einzelheiten finden Sie unter Get DICOMImport Job in AWS SDK for Python (Boto3) API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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-Details finden Sie unter Get DICOMImport Job in AWS SDK for SAP ABAP API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Importaufträge auflisten

Verwenden Sie die `ListDICOMImportJobs` Aktion, um Importaufträge aufzulisten, die für einen bestimmten HealthImaging Datenspeicher erstellt wurden. Die folgenden Menüs enthalten ein Verfahren für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `ListDICOMImportJobs` in der HealthImaging AWS-API-Referenz.

> **Note**
>
> Importaufträge werden 90 Tage lang in der Auftragsliste aufbewahrt und anschließend archiviert.

Um Importaufträge aufzulisten

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Details zum Datenspeicher wird geöffnet. Die Registerkarte Bildsätze ist standardmäßig ausgewählt.
3. Wählen Sie die Registerkarte Importe, um alle zugehörigen Importaufträge aufzulisten.

AWS CLI und SDKs

CLI

AWS CLI

So listen Sie DICOM-Importaufträge auf

Im folgenden Codebeispiel für `list-dicom-import-jobs` werden die DICOM-Importaufträge aufgelistet.

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

Ausgabe:

```
{
    "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"
        }
```

```
    ]
}
```

Weitere Informationen finden Sie unter Auflisten von Importaufträgen im AWS HealthImaging Entwicklerhandbuch.

- Einzelheiten zur API finden Sie unter DICOMImportJobs auflisten in der AWS CLI Befehlsreferenz.

Java

SDK für 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<>();
   }
```

- Einzelheiten zur API finden Sie unter DICOMImportJobs in der AWS SDK for Java 2.x API-Referenz auflisten.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

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

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

  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: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
```

```
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- Einzelheiten zur API finden Sie unter DICOMImportJobs auflisten in der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- API-Einzelheiten finden Sie unter API-Referenz zum DICOMImportAuflisten von Aufträgen in AWS SDK for Python (Boto3).

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        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.
```

- Einzelheiten zur API finden Sie in der Referenz zum DICOMImportAuflisten von Aufträgen im AWS SDK für SAP ABAP API.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Zugreifen auf Bilddatensätze mit AWS HealthImaging

Der Zugriff auf medizinische Bilddaten in AWS beinhaltet in HealthImaging der Regel die Suche nach einem Bilddatensatz mit einem eindeutigen Schlüssel und das Abrufen der zugehörigen Metadaten und Bildrahmen (Pixeldaten).

> **Wichtig**
>
> HealthImaging Verarbeitet während des Imports die Binärdateien (`.dcm`Dateien) der DICOM-Instanz und wandelt sie in Bilddatensätze um. Verwenden Sie HealthImaging Cloud-native Aktionen (APIs), um Datenspeicher und Bilddatensätze zu verwalten. Verwenden Sie HealthImaging die Darstellung von DICOMweb Diensten, um DICOMweb Antworten zurückzugeben.

In den folgenden Themen wird erklärt, wie Sie HealthImaging Cloud-native Aktionen in den AWS-Managementkonsole, und verwenden AWS CLI, AWS SDKs um Bilddatensätze zu durchsuchen und die zugehörigen Eigenschaften, Metadaten und Bildrahmen abzurufen.

Themen

- Bilddatensätze verstehen
- Suche nach Bilddatensätzen
- Eigenschaften von Bilddatensätzen abrufen
- Metadaten von Bilddatensätzen abrufen
- Pixeldaten des Bildsatzes abrufen

Bilddatensätze verstehen

Bilddatensätze ähneln einer AWS DICOM-Serie und dienen als Grundlage für AWS HealthImaging. Bilddatensätze werden erstellt, wenn Sie Ihre DICOM-Daten in importieren. HealthImaging Der Dienst versucht, importierte P10-Daten gemäß der DICOM-Hierarchie von Studie, Serie und Instanz zu organisieren.

Bilddatensätze wurden aus den folgenden Gründen eingeführt:

- Support Sie flexibel APIs eine Vielzahl von Workflows für die medizinische Bildgebung (klinisch und nichtklinisch).
- Stellen Sie einen Mechanismus zur dauerhaften Speicherung und zum Abgleich doppelter und inkonsistenter Daten bereit. Importierte P10-Daten, die mit primären Bilddatensätzen, die sich bereits im A-Speicher befinden, in Konflikt geraten, werden als nicht primäre Daten gespeichert. Nach der Lösung von Metadatenkonflikten können diese Daten als primär eingestuft werden.
- Maximieren Sie die Patientensicherheit, indem Sie nur verwandte Daten gruppieren.
- Ermutigen Sie dazu, Daten zu bereinigen, um Inkonsistenzen besser sichtbar zu machen. Weitere Informationen finden Sie unter Ändern von Bilddatensätzen.

> **Wichtig**
>
> Die klinische Verwendung von DICOM-Daten vor ihrer Bereinigung kann zu Schäden für den Patienten führen.

Die folgenden Menüs beschreiben Bilddatensätze ausführlicher und enthalten Beispiele und Diagramme, die Ihnen helfen, ihre Funktionalität und ihren Zweck in zu verstehen. HealthImaging

Was ist ein Bildsatz?

Ein Bilddatensatz ist ein AWS Konzept, das einen abstrakten Gruppierungsmechanismus zur Optimierung verwandter medizinischer Bilddaten definiert, der einer DICOM-Serie sehr ähnlich ist. Wenn Sie Ihre DICOM P10-Bilddaten in einen HealthImaging AWS-Datenspeicher importieren, werden sie in Bildsätze umgewandelt, die aus Metadaten und Bildrahmen (Pixeldaten) bestehen.

> **Note**
>
> Die Metadaten des Bildsatzes sind normalisiert. Mit anderen Worten, ein gemeinsamer Satz von Attributen und Werten wird Elementen auf Patienten-, Studien- und Serienebene zugeordnet, die im Register der DICOM-Datenelemente aufgeführt sind. HealthImaging verwendet die folgenden DICOM-Elemente, wenn eingehende DICOM P10-Objekte zu Bilddatensätzen gruppiert werden.

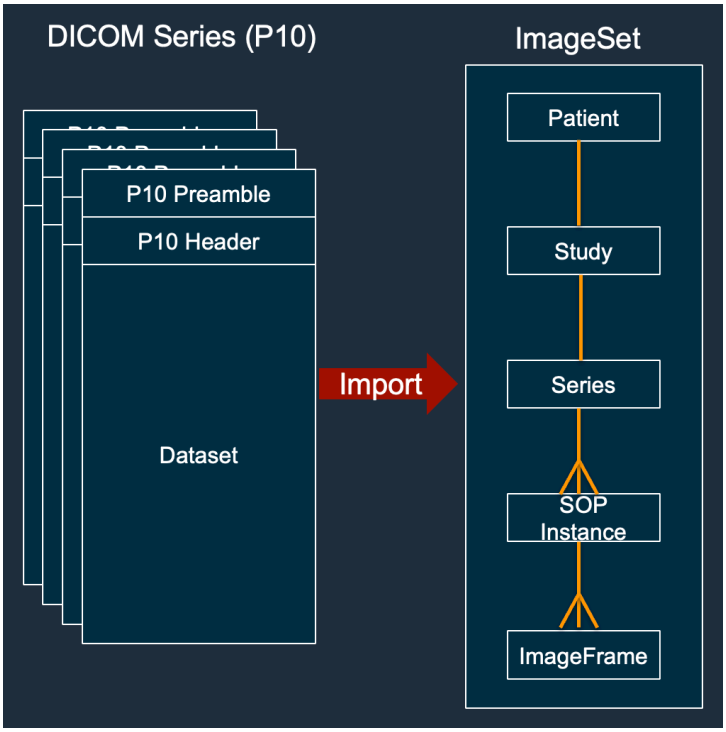
DICOM-Elemente, die für die Erstellung von Bilddatensätzen verwendet werden

Elementname	Element-Tag
Elemente auf Studienebene	
`Study Date`	`(0008,0020)`
`Accession Number`	`(0008,0050)`
`Patient ID`	`(0010,0020)`
`Study Instance UID`	`(0020,000D)`
`Study ID`	`(0020,0010)`
Elemente auf Serienebene	
`Series Instance UID`	`(0020,000E)`
`Series Number`	`(0020,0011)`

Während des Imports behalten einige Bilddatensätze ihre ursprüngliche Übertragungssyntaxkodierung bei, während andere standardmäßig in verlustfreies High-Throughput-JPEG 2000 (HTJ2K) transkodiert werden. Wenn ein Bilddatensatz in HTJ2 K codiert ist, muss er vor der Anzeige dekodiert werden. Weitere Informationen erhalten Sie unter Unterstützte Übertragungssyntaxen und Bibliotheken zur Dekodierung von Bildrahmen. Bildframes (Pixeldaten) sind in JPEG 2000 (HTJ2K) mit hohem Durchsatz codiert und müssen vor der Anzeige dekodiert werden.

Bilddatensätze sind AWS Ressourcen, daher werden ihnen Amazon-Ressourcennamen (ARNs) zugewiesen. Sie können mit bis zu 50 Schlüssel-Wert-Paaren gekennzeichnet werden und ihnen wird über IAM eine rollenbasierte Zugriffskontrolle (RBAC) und eine attributebasierte Zugriffskontrolle (ABAC) gewährt. Darüber hinaus werden Bilddatensätze versioniert, sodass alle Änderungen erhalten bleiben und auf frühere Versionen zugegriffen werden kann.

Der Import von DICOM P10-Daten führt zu Bilddatensätzen, die DICOM-Metadaten und Bildrahmen für eine oder mehrere SOP-Instanzen (Service Object Pair) in derselben DICOM-Serie enthalten.



Note

DICOM-Importaufträge:

- Erstellen Sie immer neue Bildsätze oder erhöhen Sie die Version vorhandener Bildsätze.
- Deduplizieren Sie den SOP-Instanzspeicher nicht. Bei jedem Import derselben SOP-Instanz wird zusätzlicher Speicherplatz als neuer, nicht primärer Imagesatz oder als inkrementierte Version eines vorhandenen primären Image-Sets verwendet.
- Organisieren Sie SOP-Instanzen mit konsistenten, widersprüchlichen Metadaten automatisch als primäre Bilddatensätze, die Instanzen mit konsistenten Metadaten für Patienten, Studien und Serien enthalten.
  - Wenn die Instanzen, aus denen eine DICOM-Serie besteht, in zwei oder mehr Importjobs importiert werden und die Instanzen nicht mit Instanzen kollidieren, die sich bereits im Datenspeicher befinden, werden alle Instanzen in einem primären Image-Set organisiert.
- Erstellen Sie nicht primäre Bilddatensätze, die DICOM P10-Daten enthalten, die mit bereits im Datenspeicher vorhandenen primären Bilddatensätzen in Konflikt stehen.
- Behalten Sie die zuletzt empfangenen Daten als neueste Version eines primären Bildsatzes bei.

- Wenn es sich bei den Instanzen, aus denen eine DICOM-Serie besteht, um primäre Bilddatensätze handelt und eine Instanz erneut importiert wird, wird die neue Kopie in den primären Imagesatz eingefügt, und die Version wird inkrementiert.

Wie sehen Metadaten von Bilddatensätzen aus?

Verwenden Sie die `GetImageSetMetadata` Aktion, um Bildsatz-Metadaten abzurufen. Die zurückgegebenen Metadaten sind mit komprimiert`gzip`, sodass Sie sie vor der Anzeige entpacken müssen. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen abrufen.

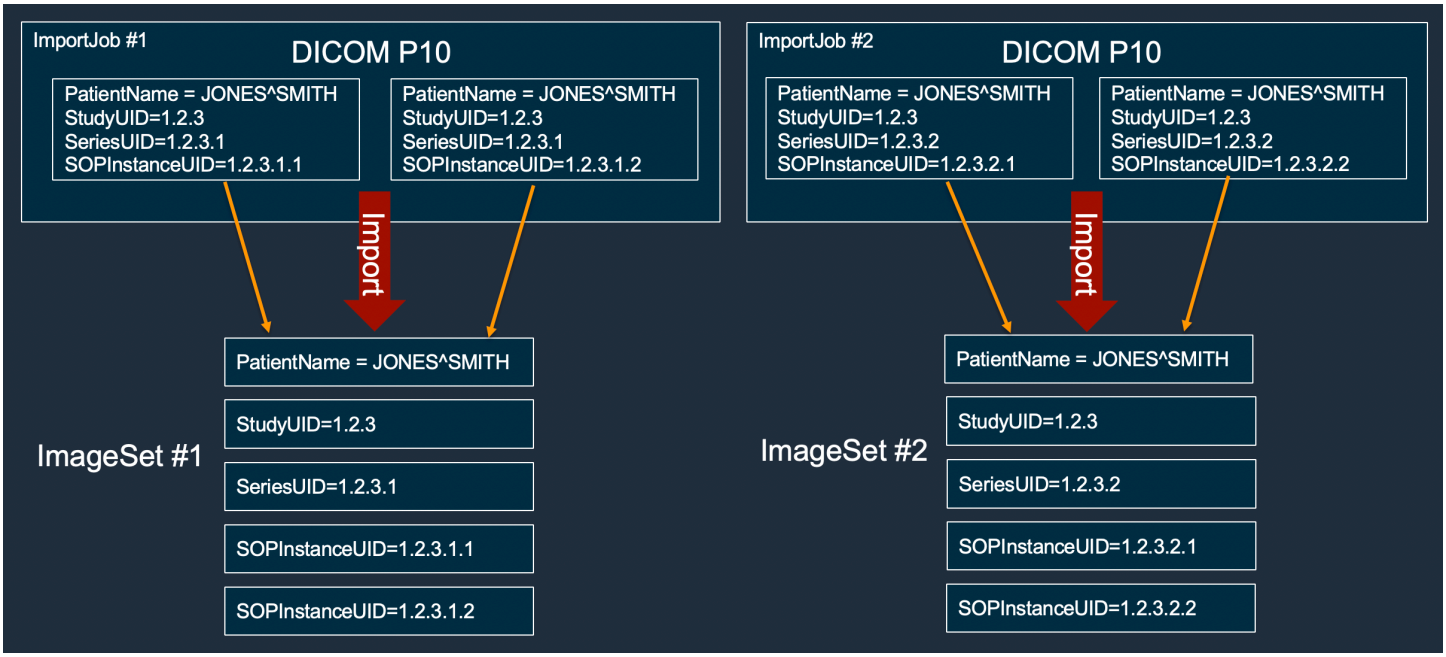
Das folgende Beispiel zeigt die Struktur von Bilddatensatz-Metadaten im JSON-Format.

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

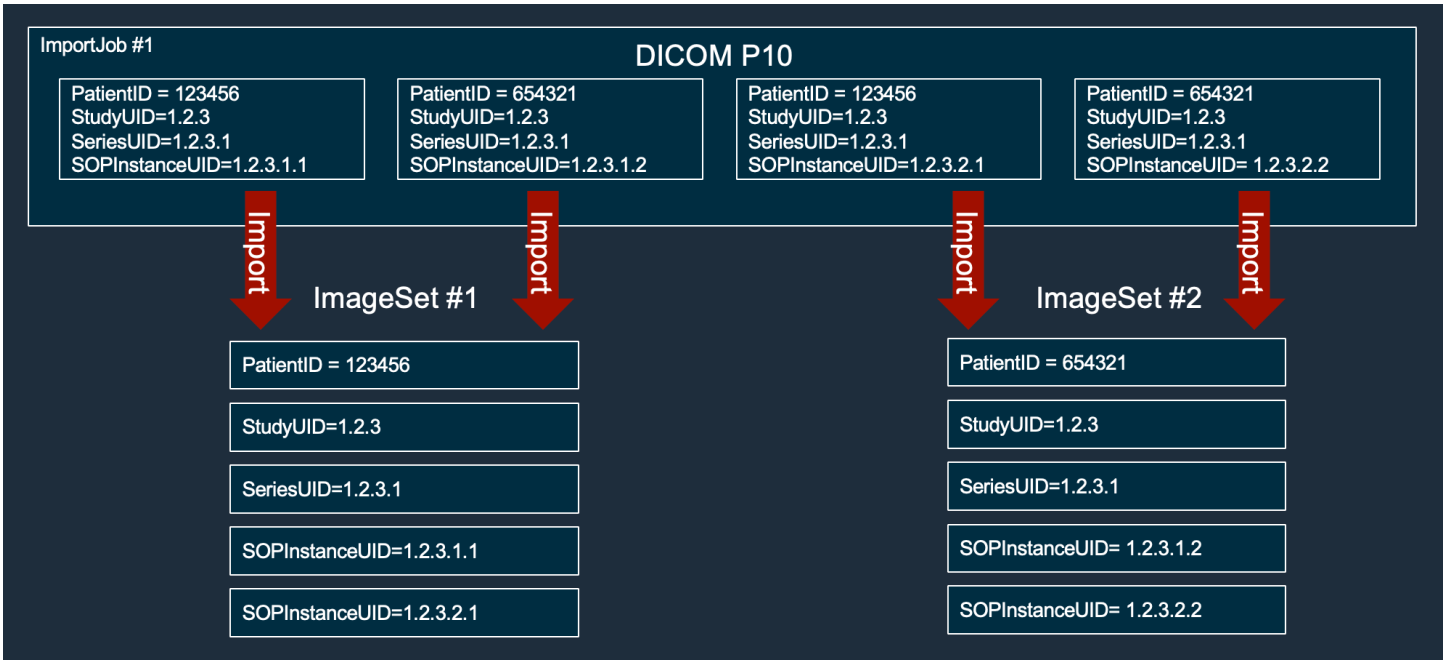
Beispiel für die Erstellung eines Bildsatzes: mehrere Importaufträge

Das folgende Beispiel zeigt, dass bei mehreren Importaufträgen immer neue Bilddatensätze erstellt und niemals zu bestehenden hinzugefügt werden.



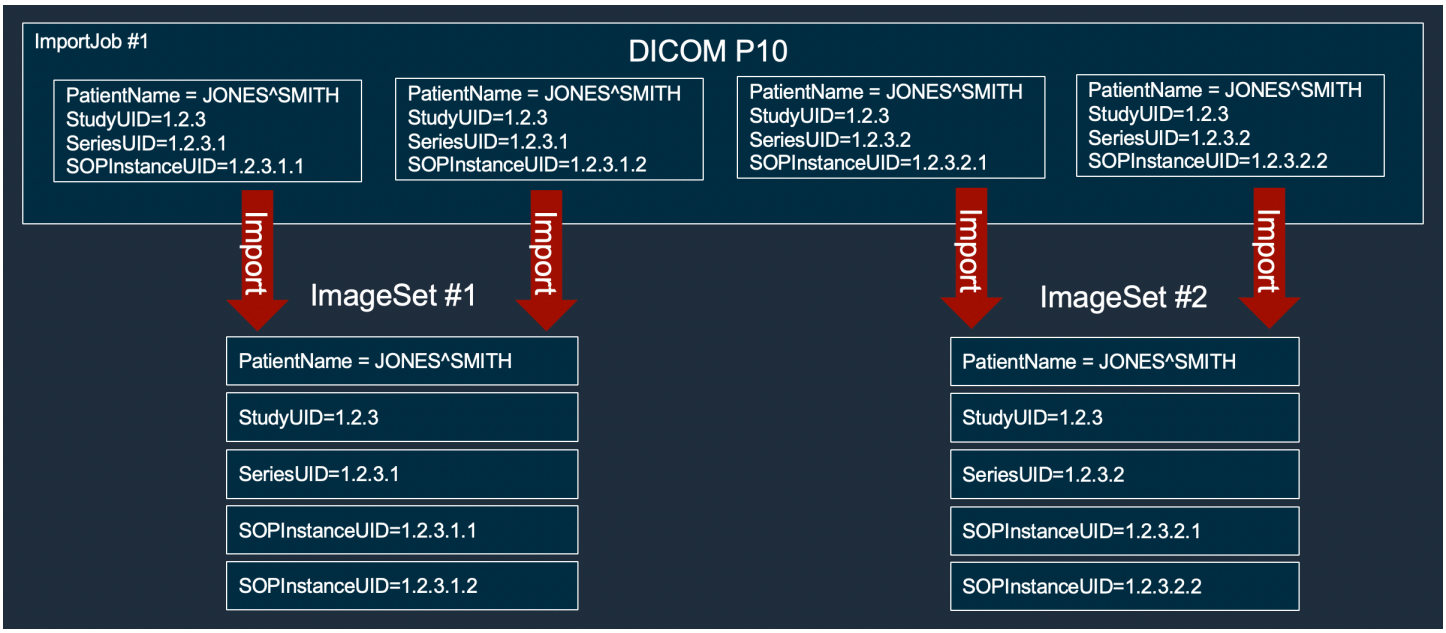
Beispiel für die Erstellung von Bilddatensätzen: Einzelner Importauftrag mit zwei Varianten

Das folgende Beispiel zeigt einen einzelnen Importauftrag, der nicht zu einem einzigen Bilddatensatz zusammengeführt werden könnte, da die Instanzen 1 und 3 einen anderen Patient haben IDs als die Instanzen 2 und 4. Um dieses Problem zu lösen, können Sie die `UpdateImageSetMetadata` Aktion verwenden, um den Konflikt zwischen der Patienten-ID und dem vorhandenen primären Bilddatensatz zu lösen. Nachdem die Konflikte gelöst wurden, können Sie die `CopyImageSet` Aktion mit dem Argument verwenden, `--promoteToPrimary` um den Bilddatensatz zum primären Bilddatensatz hinzuzufügen.



Beispiel für die Erstellung eines Bildsatzes: Einzelimportjob mit Optimierung

Das folgende Beispiel zeigt einen einzelnen Importauftrag, bei dem zwei Bilddatensätze erstellt werden, um den Durchsatz zu verbessern, obwohl die Patientennamen übereinstimmen.



Suche nach Bilddatensätzen

Verwenden Sie die `SearchImageSets` Aktion, um Suchanfragen für alle Bildsätze in einem `ACTIVE` HealthImaging Datenspeicher auszuführen. Die folgenden Menüs enthalten ein Verfahren für das

AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `SearchImageSets`in der HealthImaging AWS-API-Referenz.

> **Note**
>
> Beachten Sie bei der Suche nach Bilddatensätzen die folgenden Punkte.
>
> - `SearchImageSets`akzeptiert einen einzelnen Suchabfrageparameter und gibt eine paginierte Antwort aller Bilddatensätze zurück, die die entsprechenden Kriterien erfüllen. Alle Abfragen nach einem Zeitraum müssen als `(lowerBound, upperBound)` eingegeben werden.
> - `SearchImageSets`Verwendet das `updatedAt` Feld standardmäßig für die Sortierung in absteigender Reihenfolge vom neuesten zum ältesten.
> - Wenn Sie Ihren Datenspeicher mit einem kundeneigenen AWS KMS Schlüssel erstellt haben, müssen Sie Ihre AWS KMS Schlüsselrichtlinie aktualisieren, bevor Sie mit Bilddatensätzen interagieren können. Weitere Informationen finden Sie unter Einen vom Kunden verwalteten Schlüssel erstellen.

So suchen Sie nach Bilddatensätzen

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

> **Note**
>
> Die folgenden Verfahren zeigen, wie Bilddatensätze mithilfe der `Updated at` Eigenschaftsfilter `Series Instance UID` und durchsucht werden.

Series Instance UID

Suchen Sie mit dem **`Series Instance UID`** Eigenschaftenfilter nach Bilddatensätzen

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

Die Seite mit den Datenspeicher-Details wird geöffnet, und die Registerkarte Bildsätze ist standardmäßig ausgewählt.

3. Wählen Sie das Eigenschaftenfiltermenü und wählen Sie`Series Instance UID`.
4. Geben Sie im Feld Wert für Suche eingeben die gewünschte Serien-Instance-UID ein (fügen Sie sie ein).

> **Note**
>
> Die UID-Werte der Series Instance müssen mit den Werten identisch sein, die in der Registry of DICOM Unique Identifiers () aufgeführt sind. UIDs Beachten Sie, dass die Anforderungen eine Reihe von Zahlen beinhalten, die mindestens einen Punkt dazwischen enthalten. Perioden sind zu Beginn oder am Ende einer Series Instance nicht zulässig UIDs. Buchstaben und Leerzeichen sind nicht zulässig. Seien Sie also vorsichtig beim Kopieren und Einfügen. UIDs

5. Wählen Sie das Menü „Datumsbereich“, wählen Sie einen Zeitraum für die Series-Instance-UID und wählen Sie „Anwenden“.
6. Wählen Sie Search (Suchen) aus.

   Serieninstanzen UIDs , die in den ausgewählten Datumsbereich fallen, werden standardmäßig in der neuesten Reihenfolge zurückgegeben.

Updated at

Suchen Sie mithilfe des **`Updated at`** Eigenschaftenfilters nach Bilddatensätzen

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Datenspeicher-Details wird geöffnet, und die Registerkarte Bildsätze ist standardmäßig ausgewählt.
3. Wählen Sie das Eigenschaftenfiltermenü und wählen Sie`Updated at`.
4. Wählen Sie das Menü „Datumsbereich“, wählen Sie einen Bilddatumsbereich aus und wählen Sie „Anwenden“.
5. Wählen Sie Search (Suchen) aus.

Bilddatensätze, die in den ausgewählten Datumsbereich fallen, werden standardmäßig in der neuesten Reihenfolge zurückgegeben.

AWS CLI und SDKs

C++

SDK für C++

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
//! 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;
}
```

Anwendungsfall 1: EQUAL-Operator.

```
        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;
            }
        }
```

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

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

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

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

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

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

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

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

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

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

```
            }
        }
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
        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;
            }
        }
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
        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;
            }
        }
```

- Einzelheiten zur API finden Sie in der API-Referenz. SearchImageSetsAWS SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: So suchen Sie nach Bilddatensätzen mit einem EQUAL-Operator

Im folgenden Codebeispiel für `search-image-sets` wird der EQUAL-Operator verwendet, um Bildsätze auf der Grundlage eines bestimmten Werts zu durchsuchen.

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

Inhalt von `search-criteria.json`

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

Ausgabe:

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

Beispiel 2: Um Bilddatensätze mit einem BETWEEN-Operator unter Verwendung von DICOMStudy Datum und DICOMStudy Uhrzeit zu durchsuchen

Im folgenden Codebeispiel für `search-image-sets` wird nach Bildsätzen mit DICOM-Studien gesucht, die zwischen dem 1. Januar 1990 (12:00 Uhr) und dem 1. Januar 2023 (12:00 Uhr) generiert wurden.

Hinweis: Die DICOMStudy Uhrzeit ist optional. Wenn der Parameter nicht vorhanden ist, ist 12:00 Uhr (Beginn des Tages) der Zeitwert für die Datumsangaben, die für die Filterung bereitgestellt werden.

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

Inhalt von `search-criteria.json`

```
{
    "filters": [{
        "values": [{
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "19900101",
                "DICOMStudyTime": "000000"
```

```
            }
        },
        {
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "20230101",
                "DICOMStudyTime": "000000"
            }
        }],
        "operator": "BETWEEN"
    }]
}
```

Ausgabe:

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

Beispiel 3: So suchen Sie Bildsätze mit einem BETWEEN-Operator mithilfe von createdAt (Zeitstudien wurden bisher fortgeführt)

Im folgenden `search-image-sets` Codebeispiel wird nach Bilddatensätzen gesucht, bei denen DICOM-Studien HealthImaging zwischen den Zeitbereichen in der UTC-Zeitzone persistiert wurden.

Hinweis: Geben Sie „createdAt“ in dem im Beispiel aufgeführten Format an („1985-04-12T23:20:50.52Z“).

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

Inhalt von `search-criteria.json`

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

Ausgabe:

```
{
    "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"
```

```
    }]
}
```

Beispiel 4: Um Bilddatensätze mit einem EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt zu durchsuchen und die Antwort in ASC-Reihenfolge im Feld updatedAt zu sortieren

Das folgende `search-image-sets` Codebeispiel sucht nach Bilddatensätzen mit einem EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiert die Antwort in ASC-Reihenfolge im Feld updatedAt.

Hinweis: Geben Sie „updatedAt“ in dem im Beispiel aufgeführten Format an („1985-04-12T23:20:50.52Z“).

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

Inhalt von `search-criteria.json`

```
{
    "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"
    }
}
```

Ausgabe:

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

Weitere Informationen finden Sie unter Suchen von Bilddatensätzen im Entwicklerhandbuch.AWS HealthImaging

- Einzelheiten zur API finden Sie SearchImageSetsin der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
    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;
    }
```

Anwendungsfall 1: EQUAL-Operator.

```
        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();
        }
```

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

```
        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();
        }
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
        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();
        }
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
        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();
        }
```

- Einzelheiten zur API finden Sie in der API-Referenz. SearchImageSetsAWS SDK for Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
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;
};
```

Anwendungsfall 1: EQUAL-Operator.

```
  const datastoreId = "12345678901234567890123456789012";

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

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

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

```
  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);
}
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
const datastoreId = "1234567890123456789012345678901 2";

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

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

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
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);
}
```

- Einzelheiten zur API finden Sie in der API-Referenz. SearchImageSetsAWS SDK für JavaScript

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
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
```

Anwendungsfall 1: EQUAL-Operator.

```
        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}")
```

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

```
        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}"
        )
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
        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}"
        )
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
        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}")
```

MedicalImagingWrapper Der folgende Code instanziiert das Objekt.

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

- Einzelheiten zur API finden Sie SearchImageSetsin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie SearchImageSetsin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Eigenschaften von Bilddatensätzen abrufen

Verwenden Sie die `GetImageSet` Aktion, um Eigenschaften für einen bestimmten Bildsatz in zurückzugeben HealthImaging. Die folgenden Menüs enthalten eine Vorgehensweise für das AWS-

Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `GetImageSet`in der HealthImaging AWS-API-Referenz.

> **Note**
>
> Standardmäßig HealthImaging gibt AWS Eigenschaften für die neueste Version eines Bildsatzes zurück. Wenn Sie Eigenschaften für eine ältere Version eines Image-Sets anzeigen möchten, stellen Sie `versionId` dies Ihrer Anfrage bei.
> Verwenden Sie `GetDICOMInstance` HealthImaging die Darstellung eines DICOMweb Dienstes, um eine `.dcm` DICOM-Instanz-Binärdatei (Datei) zurückzugeben. Weitere Informationen finden Sie unter Abrufen einer DICOM-Instanz von HealthImaging.

So rufen Sie Bildsatzeigenschaften ab

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Datenspeicher-Details wird geöffnet, und die Registerkarte Bildsätze ist standardmäßig ausgewählt.
3. Wählen Sie einen Bildsatz aus.

   Die Seite mit den Bilddatensatzdetails wird geöffnet und zeigt die Eigenschaften des Bildsatzes an.

AWS CLI und SDKs

CLI

AWS CLI

So rufen Sie Bildsatzeigenschaften ab

Im folgenden Codebeispiel für `get-image-set` werden die Eigenschaften für einen Bildsatz abgerufen.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Bilddatensatz-Eigenschaften.

- Einzelheiten zur API finden Sie GetImageSetin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie GetImageSetin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
```

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

  return response;
};
```

- Einzelheiten zur API finden Sie GetImageSetin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetImageSetin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " 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.
```

- Einzelheiten zur API finden Sie GetImageSetin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Metadaten von Bilddatensätzen abrufen

Verwenden Sie die `GetImageSetMetadata` Aktion, um Metadaten für einen bestimmten Bildsatz in abzurufen HealthImaging. Die folgenden Menüs enthalten eine Vorgehensweise für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `GetImageSetMetadata`in der HealthImaging AWS-API-Referenz.

> **Note**
>
> HealthImaging Gibt standardmäßig Metadatenattribute für die neueste Version eines Image-Sets zurück. Wenn Sie Metadaten für eine ältere Version eines Bildsatzes anzeigen möchten, geben Sie dies `versionId` in Ihrer Anfrage an.
> Bilddatensatz-Metadaten werden mit einem JSON-Objekt komprimiert `gzip` und als JSON-Objekt zurückgegeben. Daher müssen Sie das JSON-Objekt dekomprimieren, bevor Sie die normalisierten Metadaten anzeigen können. Weitere Informationen finden Sie unter Normalisierung von Metadaten.
> Wenn die Metadaten eines großen Bilddatensatzes nach dem Import noch verarbeitet werden, wird `ConflictException` möglicherweise ein 409-Wert zurückgegeben. Versuchen Sie die Anfrage nach einigen Sekunden erneut, sobald die Verarbeitung abgeschlossen ist.
> Verwenden Sie `GetDICOMInstanceMetadata` HealthImaging die Darstellung eines DICOMweb Dienstes, um Metadaten (`.json`Datei) der DICOM-Instanz zurückzugeben.

Weitere Informationen finden Sie unter Metadaten der DICOM-Instanz abrufen von HealthImaging.

Um Metadaten von Bilddatensätzen abzurufen

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Datenspeicher-Details wird geöffnet, und die Registerkarte Bildsätze ist standardmäßig ausgewählt.
3. Wählen Sie einen Bildsatz aus.

   Die Seite mit den Bilddatensatzdetails wird geöffnet, und die Metadaten des Bildsatzes werden im Bereich Bilddaten-Viewer angezeigt.

AWS CLI und SDKs

C++

SDK für C++

Hilfsfunktion zum Abrufen von Bildsatz-Metadaten.

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

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

    return outcome.IsSuccess();
}
```

Rufen Sie Bildsatz-Metadaten ohne Version ab.

```
        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;
        }
```

Rufen Sie Bildsatz-Metadaten mit Version ab.

```
        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;
        }
```

- Einzelheiten zur API finden Sie GetImageSetMetadataunter AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: So rufen Sie Bildsatz-Metadaten ohne Version ab

Im folgenden Codebeispiel für `get-image-set-metadata` werden Metadaten für einen Bildsatz abgerufen, ohne eine Version anzugeben.

Anmerkung: Der Parameter `outfile` ist erforderlich.

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

Die zurückgegebenen Metadaten werden mit GZIP komprimiert und in der Datei „studymetadata.json.gz“ gespeichert. Um den Inhalt des zurückgegebenen JSON-Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren.

Ausgabe:

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

```
}
```

Beispiel 2: So rufen Sie Bildsatz-Metadaten mit Version ab

Im folgenden Codebeispiel für `get-image-set-metadata` werden Metadaten für einen Bildsatz mit einer bestimmten Version abgerufen.

Anmerkung: Der Parameter `outfile` ist erforderlich.

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

Die zurückgegebenen Metadaten werden mit GZIP komprimiert und in der Datei „studymetadata.json.gz“ gespeichert. Um den Inhalt des zurückgegebenen JSON-Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren.

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Bilddatensatz-Metadaten.

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS CLI Befehlsreferenz.

Java

SDK für 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);
        }
    }
```

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

Hilfsfunktion zum Abrufen von Bildsatz-Metadaten.

```
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 = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

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

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

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

  return response;
};
```

Rufen Sie Bildsatz-Metadaten ohne Version ab.

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

Rufen Sie Bildsatz-Metadaten mit Version ab.

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

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

Hilfsfunktion zum Abrufen von Bildsatz-Metadaten.

```
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
```

Rufen Sie Bildsatz-Metadaten ohne Version ab.

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

Rufen Sie Bildsatz-Metadaten mit Version ab.

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

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetImageSetMetadatain AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " 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.
```

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Syntax-Metadaten übertragen

HealthImaging Behält beim Import von DICOM-Daten den ursprünglichen Wert für das Transfer-Syntaxattribut in den Metadaten des Bildsatzes bei. Die Übertragungssyntax der importierten ursprünglichen DICOM-Daten wird als gespeichert. `TransferSyntaxUID` HealthImaging verwendet`StoredTransferSyntaxUID`, um das Format anzugeben, das zur Kodierung von Bildrahmendaten im Datenspeicher verwendet wird: `1.2.840.10008.1.2.4.202` für HTJ2 K-fähige Datenspeicher (Standard) und `1.2.840.10008.1.2.4.90` für JPEG 2000-Datenspeicher mit Lossless-Unterstützung.

Pixeldaten des Bildsatzes abrufen

Ein Bildrahmen sind die Pixeldaten, die in einem Bilddatensatz vorhanden sind, um ein medizinisches 2D-Bild zu erstellen. Verwenden Sie die `GetImageFrame` Aktion, um einen HTJ2 K-codierten oder systemeigenen, verlustfreien JPEG 2000-Bildrahmen für einen bestimmten Bilddatensatz abzurufen. HealthImaging Die folgenden Menüs enthalten Codebeispiele für und. AWS CLI AWS SDKs Weitere Informationen finden Sie `GetImageFrame`in der HealthImaging AWS-API-Referenz.

Note

Beachten Sie bei der Verwendung der `GetImageFrame` Aktion die folgenden Punkte:

- HealthImaging Behält beim Import die Kodierung einiger Übertragungssyntaxen bei und transkodiert andere in HTJ2 K Lossless (Standard) oder JPEG 2000 Lossless. Die `GetImageFrame` Aktion gibt den Bildrahmen in der gespeicherten Übertragungssyntax der Instanz zurück. Während des Abrufs wird keine Transcodierung durchgeführt, um eine minimale Latenz beim Abrufen zu gewährleisten. Je nach Übertragungssyntax müssen Bildframes möglicherweise vor der Anzeige in einem Bildbetrachter dekodiert werden. Weitere Informationen erhalten Sie unter Unterstützte Übertragungssyntaxen und Bibliotheken zur Dekodierung von Bildrahmen.
- Bei Instanzen, in HealthImaging denen ein oder mehrere Bildrahmen gespeichert sind, die in der MPEG-Familie der Übertragungssyntaxen (einschließlich MPEG-4 AVC/H.264 and

HEVC/H .265) kodiert sind MPEG2, gibt die `GetImageFrame` Aktion ein Videoobjekt in der gespeicherten Übertragungssyntax zurück.

- Die Übertragungssyntax von Bildrahmen ist im Header-Antwortelement angegeben. `Content-Type HTTP` Ein in HTJ2 K kodierter Bildrahmen hat `Content-Type: image/jph header` beispielsweise. Weitere Informationen finden Sie `GetImageFrame`in der HealthImaging AWS-API-Referenz.
- Sie können auch die Darstellung HealthImaging eines DICOMweb Service verwenden`GetDICOMInstanceFrames`, um DICOM-Instance-Frames (`multipart`Anfrage) für DICOMweb -kompatible Viewer und Anwendungen abzurufen. Weitere Informationen finden Sie unter DICOM-Instanz-Frames abrufen von HealthImaging.

So rufen Sie Pixeldaten des Bildersatzes ab

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

Note

Auf Bildrahmen muss programmgesteuert zugegriffen und sie dekodiert werden, da in der kein Bildbetrachter verfügbar ist. AWS-Managementkonsole
Weitere Informationen zum Dekodieren und Anzeigen von Bildrahmen finden Sie unter. Bibliotheken zur Dekodierung von Bildrahmen

AWS CLI und SDKs

C++

SDK für 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();
}
```

- Einzelheiten zur API finden Sie GetImageFrameunter AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So rufen Sie Pixeldaten des Bildersatzes ab

Im folgenden Codebeispiel für `get-image-frame` wird ein Image-Frame abgerufen.

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

Hinweis: Dieses Codebeispiel beinhaltet keine Ausgabe, da die GetImageFrame Aktion einen Stream von Pixeldaten an die Datei imageframe.jph zurückgibt. Informationen zum Dekodieren und Anzeigen von Bildrahmen finden Sie unter K-Decodierungsbibliotheken. HTJ2

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Pixeldaten von Bilddatensätzen.

- Einzelheiten zur API finden Sie GetImageFramein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
                }
        }
```

- Einzelheiten zur API finden Sie GetImageFrameunter AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie GetImageFramein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetImageFramein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie GetImageFramein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Ändern von Bilddatensätzen mit AWS HealthImaging

Bei DICOM-Importaufträgen müssen Sie Ihre Bildsätze in der Regel aus den folgenden Gründen ändern:

- Sicherheit der Patienten
- Datenkonsistenz
- Reduzieren Sie die Lagerkosten

> **Wichtig**
>
> HealthImaging Verarbeitet während des Imports die Binärdateien (`.dcm`Dateien) der DICOM-Instanz und wandelt sie in Bilddatensätze um. Verwenden Sie HealthImaging Cloud-native Aktionen (APIs), um Datenspeicher und Bilddatensätze zu verwalten. Verwenden Sie HealthImaging die Darstellung von DICOMweb Diensten, um DICOMweb Antworten zurückzugeben.

HealthImaging bietet mehrere Cloud-native Optionen APIs , um den Prozess der Änderung von Bilddatensätzen zu vereinfachen. In den folgenden Themen wird beschrieben, wie Bilddatensätze mithilfe von AWS CLI und geändert AWS SDKs werden.

Themen

- Versionen von Bildsätzen auflisten
- Metadaten von Bilddatensätzen aktualisieren
- Kopieren eines Bilddatensatzes
- Löschen eines Bilddatensatzes

Versionen von Bildsätzen auflisten

Verwenden Sie die `ListImageSetVersions` Aktion, um den Versionsverlauf für ein Image-Set in aufzulisten HealthImaging. Die folgenden Menüs enthalten eine Vorgehensweise für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `ListImageSetVersions`in der HealthImaging AWS-API-Referenz.

> **Note**
>
> AWS HealthImaging zeichnet jede Änderung auf, die an einem Bildsatz vorgenommen wurde. Durch die Aktualisierung der Bilddatensatz-Metadaten wird eine neue Version im Image-Set-Verlauf erstellt. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen aktualisieren.

Um Versionen für einen Bildsatz aufzulisten

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Datenspeicher-Details wird geöffnet, und die Registerkarte Bildsätze ist standardmäßig ausgewählt.
3. Wählen Sie einen Bildsatz aus.

   Die Seite mit den Details zum Bildset wird geöffnet.

   Die Version des Bildsatzes wird im Abschnitt Details zum Bildsatz angezeigt.

AWS CLI und SDKs

CLI

AWS CLI

So listen Sie Versionen von Bildsätzen auf

Im folgendem Codebeispiel für `list-image-set-versions` wird der Versionsverlauf für einen Bildsatz aufgelistet.

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

Ausgabe:

```
{
    "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
        }
    ]
}
```

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Auflisten von Imageset-Versionen.

- Einzelheiten zur API finden Sie ListImageSetVersionsin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
   }
```

- Einzelheiten zur API finden Sie ListImageSetVersionsunter AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Einzelheiten zur API finden Sie ListImageSetVersionsin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie ListImageSetVersionsin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie ListImageSetVersionsin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Metadaten von Bilddatensätzen aktualisieren

Verwenden Sie die `UpdateImageSetMetadata` Aktion, um die Metadaten des Bildsatzes in AWS zu aktualisieren HealthImaging. Sie können diesen asynchronen Prozess verwenden, um Metadatenattribute von Bilddatensätzen hinzuzufügen, zu aktualisieren und zu entfernen. Dabei handelt es sich um Manifestationen von DICOM-Normalisierungselementen, die während des Imports erstellt werden. Mithilfe der `UpdateImageSetMetadata` Aktion können Sie einzelne SOP-Instanzen

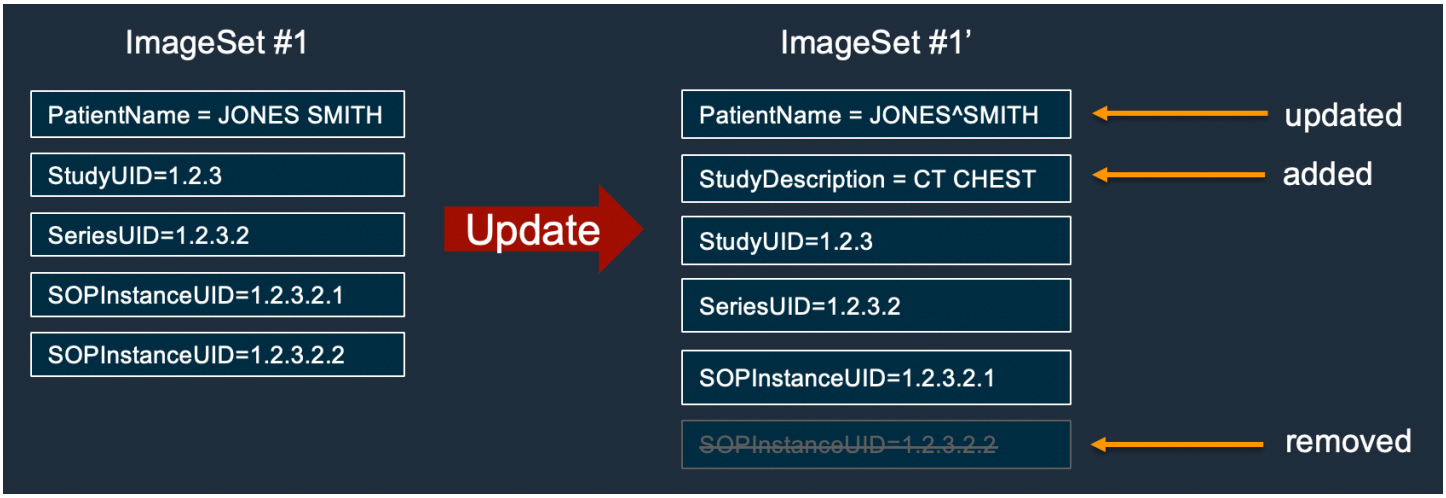
entfernen, um Bilddatensätze mit externen Systemen synchron zu halten und Bilddatensatz-Metadaten zu anonymisieren. Weitere Informationen finden Sie `UpdateImageSetMetadata`in der HealthImaging AWS-API-Referenz.

> **Note**
>
> Bei echten DICOM-Importen müssen Attribute in den Metadaten des Bildsatzes aktualisiert, hinzugefügt und entfernt werden. Beachten Sie bei der Aktualisierung von Bilddatensatz-Metadaten die folgenden Punkte:
>
> - Übergeben Sie die `--include-study-image-sets` Markierung, um alle primären Bilddatensätze zu aktualisieren, die dieselbe Study Instance UID wie der angeforderte Bilddatensatz haben. Dies ist ein atomarer Vorgang, bei dem die Versionen aller betroffenen Bilddatensätze inkrementiert werden. Hinweis: Das `--include-study-image-sets` Flag wird bei `revertToVersionId` Operationen nicht unterstützt, da Revert eine frühere Version wiederherstellt und keine Attributänderungen anwendet.
> - Durch das Aktualisieren von Bilddatensatz-Metadaten wird eine neue Version im Verlauf des Bilddatensatzes erstellt. Weitere Informationen finden Sie unter Versionen von Bildsätzen auflisten. Verwenden Sie den optionalen `revertToVersionId`Parameter, um zu einer früheren Versions-ID des Bildsatzes zurückzukehren.
> - Das Aktualisieren von Bilddatensatz-Metadaten ist ein asynchroner Prozess. Daher `imageSetState`sind `imageSetWorkflowStatus`Antwortelemente verfügbar, die den jeweiligen Status und Status eines Bilddatensatzes angeben, der gerade aktualisiert wird. Sie können keine anderen Schreibvorgänge an einem `LOCKED` Bilddatensatz ausführen.
> - Wenn die `UpdateImageSetMetadata` Aktion nicht erfolgreich ist, rufen Sie das `message`Antwortelement auf und überprüfen Sie es `common errors.`
> - DICOM-Elementbeschränkungen werden auf Metadaten-Updates angewendet. Der `force`Anforderungsparameter ermöglicht es Ihnen, Elemente von Bilddatensätzen zu aktualisieren, die nicht primär sind, wenn Sie sie überschreiben möchten. Einschränkungen für DICOM-Metadaten
> - Die Aktualisierung von UID, `force` UID und StudyInstance SOPInstance UID für primäre SeriesInstance Bilddatensätze UpdateImageSet wird nicht unterstützt.
> - Stellen Sie den `force`Anforderungsparameter ein, um den Abschluss der `UpdateImageSetMetadata` Aktion für Bilddatensätze zu erzwingen, die nicht primär sind. Die Einstellung dieses Parameters ermöglicht die folgenden Aktualisierungen eines Bilddatensatzes:

- Aktualisierung der `Tag.StudyID` Attribute `Tag.StudyInstanceUID` `Tag.SeriesInstanceUIDTag.SOPInstanceUID`,, und
- Private DICOM-Datenelemente auf Instanzebene hinzufügen, entfernen oder aktualisieren

- Um eine Instanz aus einem Image-Set zu entfernen, muss der `--force` Parameter in der `UpdateImageSetMetadata` Anfrage angegeben werden.
- Durch das Heraufstufen eines Bilddatensatzes zum primären Bilddatensatz wird die ID des Bildsatzes geändert.
- Bei der Aktualisierung eines VR=SQ-Attributs wird das gesamte Sequenzattribut aktualisiert. Diese API unterstützt keine teilweisen Aktualisierungen von Sequenzattributen.

Das folgende Diagramm zeigt Bilddatensatz-Metadaten, in denen aktualisiert werden HealthImaging.



Um Bildsatz-Metadaten zu aktualisieren

Wählen Sie eine Registerkarte, die auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: So fügen Sie ein Attribut in Bildsatz-Metadaten ein oder aktualisieren es

Im folgenden Beispiel für `update-image-set-metadata` wird ein Attribut in die Metadaten eines Bildsatzes eingefügt.

```
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
```

Inhalt von `metadata-updates.json`

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

Ausgabe:

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

Beispiel 2: So entfernen Sie ein Attribut aus den Metadaten eines Bildsatzes

Im folgenden Beispiel für `update-image-set-metadata` wird ein Attribut aus den Metadaten eines Bildsatzes entfernt.

```
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
```

Inhalt von `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "removableAttributes": "{\"SchemaVersion\":1.1,\"Study\":{\"DICOM\":
{\"StudyDescription\":\"CHEST\"}}}"
    }
}
```

Ausgabe:

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

Beispiel 3: So entfernen Sie eine Instance aus den Metadaten eines Bildsatzes

Im folgenden Beispiel für `update-image-set-metadata` wird eine Instance aus den Metadaten eines Bildsatzes entfernt.

```
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
```

Inhalt von `metadata-updates.json`

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

Ausgabe:

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

Beispiel 4: So setzen Sie einen Bildsatz auf eine frühere Version zurück

Das folgende `update-image-set-metadata` Beispiel zeigt, wie Sie einen Image-Satz auf eine frühere Version zurücksetzen können. CopyImageSet und UpdateImageSetMetadata Aktionen erstellen neue Versionen von Bilddatensätzen.

```
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"}'
```

Ausgabe:

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

Beispiel 5: So fügen Sie einer Instance ein privates DICOM-Datenelement hinzu

Das folgende Beispiel für `update-image-set-metadata` zeigt, wie ein privates Element zu einer angegebenen Instance innerhalb eines Bildsatzes hinzugefügt wird. Der DICOM-

Standard erlaubt private Datenelemente für die Übertragung von Informationen, die nicht in Standarddatenelementen enthalten sein dürfen. Mit der UpdateImageSetMetadata Aktion können Sie private Datenelemente erstellen, aktualisieren und löschen.

```
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
```

Inhalt von `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"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\": {\"DICOM\":
 {\"001910F9\": \"97\"},\"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}"
    }
}
```

Ausgabe:

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

Beispiel 6: So aktualisieren Sie ein privates DICOM-Datenelement in einer Instance

Das folgende Beispiel für `update-image-set-metadata` zeigt, wie der Wert eines privates Datenelements, das zu einer Instance eines Bildsatzes gehört, aktualisiert wird.

```
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
```

Inhalt von `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"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\": {\"DICOM\":
 {\"00091001\": \"GE_GENESIS_DD\"}}}}}}"
    }
}
```

Ausgabe:

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

Beispiel 7: Um eine SOPInstance UID mit dem Force-Parameter zu aktualisieren

Das folgende `update-image-set-metadata` Beispiel zeigt, wie eine SOPInstance UID aktualisiert wird, indem der Force-Parameter verwendet wird, um die DICOM-Metadatenbeschränkungen zu überschreiben.

```
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
```

Inhalt von `metadata-updates.json`

```
{
    "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\"}}}}}}}"
    }
}
```

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Aktualisieren von Bilddatensatz-Metadaten.

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain der AWS CLI Befehlsreferenz.

Java

SDK für 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;
       }
   }
```

Anwendungsfall 1: Fügen Sie ein Attribut ein oder aktualisieren Sie es.

```
               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);
```

Anwendungsfall 2: Entfernen Sie ein Attribut.

```
              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);
```

Anwendungsfall 3: Entfernen Sie eine Instance.

```
                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);
```

Anwendungsfall 4: Kehren Sie zu einer früheren Version zurück.

```
                // 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);
```

- Einzelheiten zur API finden Sie UpdateImageSetMetadataunter AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

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

Anwendungsfall 1: Fügen Sie ein Attribut ein oder aktualisieren Sie es und erzwingen Sie das Update.

```
    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,
);
```

Anwendungsfall 2: Entfernen Sie ein Attribut.

```
// 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,
);
```

Anwendungsfall 3: Entfernen Sie eine Instance.

```
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,
);
```

Anwendungsfall 4: Kehren Sie zu einer früheren Version zurück.

```
const updateMetadata = {
  revertToVersionId: "1",
};

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

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

Anwendungsfall 1: Fügen Sie ein Attribut ein oder aktualisieren Sie es.

```
            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
            )
```

Anwendungsfall 2: Entfernen Sie ein Attribut.

```
            # 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
        )
```

Anwendungsfall 3: Entfernen Sie eine Instance.

```
        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
        )
```

Anwendungsfall 4: Kehren Sie zu einer früheren Version zurück.

```
        metadata = {"revertToVersionId": "1"}

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

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Mithilfe von, und können Sie SOP-Instanzen zwischen Bilddatensätzen verschieben, Konflikte mit Metadatenelementen lösen und Instanzen zu den primären Bilddatensätzen hinzufügen oder `CopyImageSet` daraus `UpdateImageSetMetadata` entfernen. `DeleteImageSet` APIs

Mit der `DeleteImageSet` Aktion können Sie einen Bildsatz aus der primären Sammlung entfernen.

Um die Metadaten eines primären Bilddatensatzes zu aktualisieren

1. Verwenden Sie die CopyImageSet Aktion, um einen nicht-primären Bildsatz zu erstellen, der eine Kopie des primären Bildsatzes ist, den Sie ändern möchten. Nehmen wir an, dies wird `103785414bc2c89330f7ce51bbd13f7a` als ID des nicht primären Bilddatensatzes zurückgegeben.

```
aws medical-imaging copy-image-set --datastore-id
a8d19e7875e1532d9b5652f6b25e12c9 --source-image-set-id
0778b83b36eced0b76752bfe32192fb7 --copy-image-set-information
'{"sourceImageSet": {"latestVersionId": "1" }}' --region us-west-2
```

2. Verwenden Sie die UpdateImageSetMetadata Aktion, um Änderungen am nichtprimären Bilddatensatz vorzunehmen. (`103785414bc2c89330f7ce51bbd13f7a`) Zum Beispiel das Ändern der PatientenID.

```
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\"}}}"
    }
  }'
```

3. Löschen Sie den primären Bilddatensatz, den Sie ändern.

```
aws medical-imaging delete-image-set --datastore-
          id a8d19e7875e1532d9b5652f6b25e12c9 --image-set-
          id 0778b83b36eced0b76752bfe32192fb7
```

4. Verwenden Sie die CopyImageSet Aktion mit dem Argument`--promoteToPrimary`, um den aktualisierten Bildsatz zur primären Sammlung hinzuzufügen.

```
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. Löschen Sie den nicht primären Bildsatz.

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

Um einen Bildsatz, der nicht primär ist, primär zu machen

1. Verwenden Sie die UpdateImageSetMetadata Aktion, um Konflikte mit vorhandenen primären Bilddatensätzen zu lösen.

```
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. Wenn die Konflikte gelöst sind, verwenden Sie die CopyImageSet Aktion mit dem Argument, `--promoteToPrimary` um den Bilddatensatz zur Sammlung des primären Bilddatensatzes hinzuzufügen.

```
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
```

3. Nachdem Sie bestätigt haben, dass die CopyImageSet Aktion erfolgreich war, löschen Sie den Quell-Imagedatensatz.

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

Kopieren eines Bilddatensatzes

Verwenden Sie die `CopyImageSet` Aktion, um einen Bildsatz zu kopieren HealthImaging. Sie verwenden diesen asynchronen Prozess, um den Inhalt eines Bilddatensatzes in einen neuen oder vorhandenen Bildsatz zu kopieren. Sie können in einen neuen Bilddatensatz kopieren, um

einen Bilddatensatz aufzuteilen oder um eine separate Kopie zu erstellen. Sie können auch in einen vorhandenen Bilddatensatz kopieren, um zwei Bilddatensätze zusammenzuführen. Weitere Informationen finden Sie `CopyImageSet`in der HealthImaging AWS-API-Referenz.

> **Note**
>
> Beachten Sie bei der Verwendung der `CopyImageSet` Aktion die folgenden Punkte:
>
> - Die `CopyImageSet` Aktion erstellt einen neuen Bildsatz oder eine neue Version von`destinationImageSet`. Weitere Informationen finden Sie unter Versionen von Bildsätzen auflisten.
> - Das Kopieren ist ein asynchroner Prozess. Daher sind die Antwortelemente state (`imageSetState`) und status (`imageSetWorkflowStatus`) verfügbar, um Sie darüber zu informieren, welcher Vorgang mit einem gesperrten Bilddatensatz ausgeführt wird. Andere Schreiboperationen können für einen gesperrten Bilddatensatz nicht ausgeführt werden.
> - `CopyImageSet`erfordert, dass die SOP-Instanz UIDs innerhalb eines Bilddatensatzes eindeutig ist.
> - Sie können Teilmengen von SOP-Instanzen kopieren mit `copiableAttributes`. Auf diese Weise können Sie eine oder mehrere SOP-Instanzen von auswählen, `sourceImageSet` um sie in die `destinationImageSet` zu kopieren.
> - Wenn die `CopyImageSet` Aktion nicht erfolgreich ist, rufen Sie an `GetImageSet` und überprüfen Sie das `message`Objekt. Weitere Informationen finden Sie unter Eigenschaften von Bilddatensätzen abrufen.
> - Reale DICOM-Importe können zu mehreren Bilddatensätzen pro DICOM-Serie führen. Für die `CopyImageSet` Aktion sind konsistente Metadaten erforderlich `sourceImageSet` und `destinationImageSet` müssen vorhanden sein, sofern der optionale `force`Parameter nicht angegeben wird.
> - Stellen Sie den `force`Parameter so ein, dass der Vorgang erzwungen wird, auch wenn sich zwischen `sourceImageSet` und `destinationImageSet` inkonsistente Metadatenelemente befinden. In diesen Fällen bleiben die Metadaten für den Patienten, die Studie und die Serie unverändert in der`destinationImageSet`.

Um einen Bilddatensatz zu kopieren

Wählen Sie eine Registerkarte, die auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: So kopieren Sie einen Bildsatz ohne Ziel.

Im folgenden Beispiel für `copy-image-set` wird eine doppelte Kopie eines Bildersatzes ohne Ziel erstellt.

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

Ausgabe:

```
{
    "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"
}
```

Beispiel 2: So kopieren Sie einen Bildersatz mit Ziel.

Im folgenden Beispiel für `copy-image-set` wird eine doppelte Kopie eines Bildersatzes mit Ziel erstellt.

```
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"} }'
```

Ausgabe:

```
{
    "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"
}
```

Beispiel 3: So kopieren Sie eine Teilmenge von Instances aus einem Quellbildersatz in einen Zielbildersatz.

Im folgenden Beispiel für `copy-image-set` wird eine DICOM-Instance aus dem Quellbildersatz in den Zielbildersatz kopiert. Der force-Parameter dient dazu, Inkonsistenzen in den Attributen auf Patienten-, Studien- und Serienebene zu überschreiben.

```
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
```

Ausgabe:

```
{
    "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"
}
```

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Kopieren eines Image-Sets.

- Einzelheiten zur API finden Sie CopyImageSetin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
        }
    }
```

Hilfsfunktion zum Erstellen kopierbarer Attribute.

```
    /**
     * 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();
    }
```

- Einzelheiten zur API finden Sie CopyImageSetin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

Hilfsfunktion zum Kopieren eines Imagesatzes.

```
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 = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
  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: 'xxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
```

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

Kopieren Sie einen Imagesatz ohne Ziel.

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

Kopieren Sie einen Imagesatz mit Ziel.

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

Kopieren Sie eine Teilmenge eines Imagesatzes mit einem Ziel und erzwingen Sie den Kopiervorgang.

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

- Einzelheiten zur API finden Sie CopyImageSetin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

Hilfsfunktion zum Kopieren eines Imagesatzes.

```
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"]
```

Kopieren Sie einen Imagesatz ohne Ziel.

```
            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,
            )
```

Kopieren Sie einen Imagesatz mit Ziel.

```
            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,
            )
```

Kopieren Sie eine Teilmenge eines Imagesatzes.

```
            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,
            )
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie CopyImageSetin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_image_set_id = '12345678901234567890123456789012345678901234567890'
        " 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.
```

- Einzelheiten zur API finden Sie CopyImageSetin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Löschen eines Bilddatensatzes

Verwenden Sie die `DeleteImageSet` Aktion, um einen Bildsatz in zu löschen HealthImaging. Die folgenden Menüs enthalten eine Vorgehensweise für das AWS-Managementkonsole und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie `DeleteImageSet`in der HealthImaging AWS-API-Referenz.

So löschen Sie einen Bildersatz

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Datenspeicher-Details wird geöffnet, und die Registerkarte Bildsätze ist standardmäßig ausgewählt.
3. Wählen Sie einen Bildsatz aus und klicken Sie auf Löschen.

   Das Modal Bildsatz löschen wird geöffnet.
4. Geben Sie die ID des Bilddatensatzes ein und wählen Sie Bildsatz löschen.

AWS CLI und SDKs

C++

SDK für 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();
}
```

- Einzelheiten zur API finden Sie DeleteImageSetunter AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So löschen Sie einen Bildersatz

Im folgenden Codebeispiel für `delete-image-set` wird ein Bildersatz gelöscht.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Löschen eines Bilddatensatzes.

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS CLI Befehlsreferenz.

Java

SDK für 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);
       }
   }
```

- Einzelheiten zur API finden Sie DeleteImageSetunter AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

```
/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie DeleteImageSetin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    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.
```

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Ressourcen mit AWS taggen HealthImaging

Sie können HealthImaging Ressourcen (Datenspeichern und Bilddatensätzen) Metadaten in Form von Tags zuweisen. Jedes Tag ist eine Bezeichnung, die aus einem benutzerdefinierten Schlüssel und Wert besteht. Mithilfe von Tags können Sie Ressourcen verwalten, identifizieren, organisieren, suchen und filtern.

> **Wichtig**
>
> Speichern Sie keine geschützten Gesundheitsinformationen (PHI), personenbezogenen Daten (PII) oder andere vertrauliche oder sensible Informationen in Tags. Tags sind nicht für private oder sensible Daten gedacht.

In den folgenden Themen wird beschrieben, wie HealthImaging Tagging-Operationen mithilfe von AWS-Managementkonsole AWS CLI, und verwendet werden. AWS SDKs Weitere Informationen finden Sie im Handbuch unter AWS Ressourcen taggen.Allgemeine AWS-Referenz

Themen

- Eine Ressource taggen
- Tags für eine Ressource auflisten
- Eine Ressource entkennzeichnen

Eine Ressource taggen

Verwenden Sie die `TagResource`Aktion, um Datenspeicher und Bildsätze in AWS zu taggen HealthImaging. In den folgenden Codebeispielen wird beschrieben, wie die `TagResource` Aktion mit AWS-Managementkonsole AWS CLI, und verwendet wird AWS SDKs. Weitere Informationen finden Sie im Allgemeine AWS-Referenz Leitfaden unter AWS Ressourcen taggen.

So taggen Sie eine Ressource

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.

2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Details zum Datenspeicher wird geöffnet.
3. Wählen Sie die Registerkarte Details.
4. Wählen Sie im Abschnitt Tags die Option Tags verwalten aus.

   Die Seite „Tags verwalten“ wird geöffnet.
5. Wählen Sie Neues Tag hinzufügen aus.
6. Geben Sie einen Schlüssel und einen Wert ein (optional).
7. Wählen Sie Änderungen speichern aus.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: So markieren Sie einen Datenspeicher

In den folgenden Codebeispielen für `tag-resource` wird ein Datenspeicher mit Tags versehen.

```
aws medical-imaging tag-resource \
  --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
  --tags '{"Deployment":"Development"}'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Beispiel 2: So taggen Sie einen Bildsatz

In den folgenden Codebeispielen für `tag-resource` wird ein Bildsatz mit Tags versehen.

```
aws medical-imaging tag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
    --tags '{"Deployment":"Development"}'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Ressourcen taggen mit AWS HealthImaging im AWS HealthImaging Entwicklerhandbuch.

- Einzelheiten zur API finden Sie TagResourcein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
       }
   }
```

- Einzelheiten zur API finden Sie TagResourcein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie TagResourcein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie TagResourcein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie TagResourcein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Tags für eine Ressource auflisten

Verwenden Sie die `ListTagsForResource`Aktion, um Tags für Datenspeicher und Bildsätze in AWS aufzulisten HealthImaging. In den folgenden Codebeispielen wird beschrieben, wie die `ListTagsForResource` Aktion mit AWS-Managementkonsole AWS CLI, und verwendet wird AWS SDKs. Weitere Informationen finden Sie im Allgemeine AWS-Referenz Leitfaden unter AWS Ressourcen taggen.

So listen Sie Tags für eine Ressource auf

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Details zum Datenspeicher wird geöffnet.
3. Wählen Sie die Registerkarte Details.

   Im Abschnitt „Tags“ werden alle Datenspeicher-Tags aufgeführt.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: So listen Sie Ressourcen-Tags für einen Datenspeicher auf

Im folgenden Codebeispiel für `list-tags-for-resource` werden die Tags für einen Datenspeicher aufgelistet.

```
aws medical-imaging list-tags-for-resource \
```

```
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"
```

Ausgabe:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

Beispiel 2: So listen Sie Ressourcen-Tags für einen Bildsatz auf

Im folgenden Codebeispiel für `list-tags-for-resource` werden die Tags für einen Bildsatz aufgelistet.

```
aws medical-imaging list-tags-for-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b"
```

Ausgabe:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

Weitere Informationen finden Sie AWS HealthImaging im AWS HealthImaging Entwicklerhandbuch unter Ressourcen taggen mit.

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie ListTagsForResourcein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Eine Ressource entkennzeichnen

Verwenden Sie die `UntagResource`Aktion, um Datenspeicher und Bildsätze in AWS HealthImaging zu entkennzeichnen. In den folgenden Codebeispielen wird beschrieben, wie die `UntagResource` Aktion mit AWS-Managementkonsole AWS CLI, und AWS SDKs verwendet wird. Weitere Informationen finden Sie im Allgemeine AWS-Referenz Leitfaden unter AWS Ressourcen taggen.

So entfernen Sie Tags von einer Ressource

Wählen Sie ein Menü, das auf Ihren Zugriffspräferenzen für AWS basiert HealthImaging.

AWS Konsole

1. Öffnen Sie die Seite Datenspeicher der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.

   Die Seite mit den Details zum Datenspeicher wird geöffnet.
3. Wählen Sie die Registerkarte Details.
4. Wählen Sie im Abschnitt Tags die Option Tags verwalten aus.

   Die Seite „Tags verwalten“ wird geöffnet.
5. Wählen Sie neben dem Tag, das Sie entfernen möchten, die Option Entfernen aus.
6. Wählen Sie Änderungen speichern aus.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: So heben Sie die Markierung eines Datenspeichers auf

Im folgenden Codebeispiel für `untag-resource` werden Tags aus einem Datenspeicher entfernt.

```
aws medical-imaging untag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
    --tag-keys '["Deployment"]'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Beispiel 2: So heben Sie das Tagging eines Bildsatzes auf

Im folgenden Codebeispiel für `untag-resource` werden Tags aus einem Bildsatz entfernt.

```
aws medical-imaging untag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
```

```
    --tag-keys '["Deployment"]'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie AWS HealthImaging im AWS HealthImaging Entwicklerhandbuch unter Ressourcen taggen mit.

- Einzelheiten zur API finden Sie UntagResourcein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
       }
   }
```

- Einzelheiten zur API finden Sie UntagResourcein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie UntagResourcein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie UntagResourcein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie UntagResourcein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie über den Link Feedback geben in der rechten Seitenleiste dieser Seite ein Codebeispiel an.

Codebeispiele für die HealthImaging Verwendung AWS SDKs

Die folgenden Codebeispiele zeigen, wie die Verwendung HealthImaging mit einem AWS Software Development Kit (SDK) funktioniert.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Codebeispiele

- Grundlegende Beispiele für die HealthImaging Verwendung AWS SDKs
  - Hallo HealthImaging
  - Aktionen zur HealthImaging Verwendung AWS SDKs
    - Verwendung CopyImageSet mit einem AWS SDK oder CLI
    - Verwendung CreateDatastore mit einem AWS SDK oder CLI
    - Verwendung DeleteDatastore mit einem AWS SDK oder CLI
    - Verwendung DeleteImageSet mit einem AWS SDK oder CLI
    - Verwendung GetDICOMImportJob mit einem AWS SDK oder CLI
    - Verwendung GetDatastore mit einem AWS SDK oder CLI
    - Verwendung GetImageFrame mit einem AWS SDK oder CLI
    - Verwendung GetImageSet mit einem AWS SDK oder CLI
    - Verwendung GetImageSetMetadata mit einem AWS SDK oder CLI
    - Verwendung ListDICOMImportJobs mit einem AWS SDK oder CLI
    - Verwendung ListDatastores mit einem AWS SDK oder CLI
    - Verwendung ListImageSetVersions mit einem AWS SDK oder CLI
    - Verwendung ListTagsForResource mit einem AWS SDK oder CLI

    - Verwendung SearchImageSets mit einem AWS SDK oder CLI
    - Verwendung StartDICOMImportJob mit einem AWS SDK oder CLI
    - Verwendung TagResource mit einem AWS SDK oder CLI
    - Verwendung UntagResource mit einem AWS SDK oder CLI
    - Verwendung UpdateImageSetMetadata mit einem AWS SDK oder CLI
- Szenarien für die HealthImaging Verwendung AWS SDKs
    - Erste Schritte mit HealthImaging Bildsätzen und Bildrahmen mithilfe eines AWS SDK
    - Kennzeichnen eines HealthImaging Datenspeichers mithilfe eines SDK AWS
    - Taggen eines HealthImaging Bilddatensatzes mithilfe eines SDK AWS

Grundlegende Beispiele für die HealthImaging Verwendung AWS SDKs

Die folgenden Codebeispiele zeigen, wie die Grundlagen von AWS HealthImaging with verwendet AWS SDKs werden.

Beispiele

- Hallo HealthImaging
- Aktionen zur HealthImaging Verwendung AWS SDKs
    - Verwendung CopyImageSet mit einem AWS SDK oder CLI
    - Verwendung CreateDatastore mit einem AWS SDK oder CLI
    - Verwendung DeleteDatastore mit einem AWS SDK oder CLI
    - Verwendung DeleteImageSet mit einem AWS SDK oder CLI
    - Verwendung GetDICOMImportJob mit einem AWS SDK oder CLI
    - Verwendung GetDatastore mit einem AWS SDK oder CLI
    - Verwendung GetImageFrame mit einem AWS SDK oder CLI
    - Verwendung GetImageSet mit einem AWS SDK oder CLI
    - Verwendung GetImageSetMetadata mit einem AWS SDK oder CLI
    - Verwendung ListDICOMImportJobs mit einem AWS SDK oder CLI
    - Verwendung ListDatastores mit einem AWS SDK oder CLI
    - Verwendung ListImageSetVersions mit einem AWS SDK oder CLI

- Verwendung ListTagsForResource mit einem AWS SDK oder CLI
- Verwendung SearchImageSets mit einem AWS SDK oder CLI
- Verwendung StartDICOMImportJob mit einem AWS SDK oder CLI
- Verwendung TagResource mit einem AWS SDK oder CLI
- Verwendung UntagResource mit einem AWS SDK oder CLI
- Verwendung UpdateImageSetMetadata mit einem AWS SDK oder CLI

Hallo HealthImaging

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung von HealthImaging beginnen.

C++

SDK für C++

Code für die CMake Datei CMake Lists.txt.

```
# 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})
```

Code für die Quelldatei „hello_health_imaging.cpp“.

```
#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;
}
```

- Einzelheiten zur API finden Sie ListDatastoresunter AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"))
```

- Einzelheiten zur API finden Sie ListDatastoresin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktionen zur HealthImaging Verwendung AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie einzelne HealthImaging Aktionen mit ausführen AWS SDKs. Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes finden.

Diese Auszüge rufen die HealthImaging API auf und sind Codeauszüge aus größeren Programmen, die im Kontext ausgeführt werden müssen. Sie können Aktionen im Kontext unter Szenarien für die HealthImaging Verwendung AWS SDKs anzeigen.

Die folgenden Beispiele enthalten nur die am häufigsten verwendeten Aktionen. Eine vollständige Liste finden Sie in der AWS HealthImaging -API-Referenz.

Beispiele

- Verwendung CopyImageSet mit einem AWS SDK oder CLI
- Verwendung CreateDatastore mit einem AWS SDK oder CLI
- Verwendung DeleteDatastore mit einem AWS SDK oder CLI
- Verwendung DeleteImageSet mit einem AWS SDK oder CLI
- Verwendung GetDICOMImportJob mit einem AWS SDK oder CLI
- Verwendung GetDatastore mit einem AWS SDK oder CLI
- Verwendung GetImageFrame mit einem AWS SDK oder CLI
- Verwendung GetImageSet mit einem AWS SDK oder CLI
- Verwendung GetImageSetMetadata mit einem AWS SDK oder CLI
- Verwendung ListDICOMImportJobs mit einem AWS SDK oder CLI
- Verwendung ListDatastores mit einem AWS SDK oder CLI
- Verwendung ListImageSetVersions mit einem AWS SDK oder CLI
- Verwendung ListTagsForResource mit einem AWS SDK oder CLI
- Verwendung SearchImageSets mit einem AWS SDK oder CLI
- Verwendung StartDICOMImportJob mit einem AWS SDK oder CLI
- Verwendung TagResource mit einem AWS SDK oder CLI
- Verwendung UntagResource mit einem AWS SDK oder CLI
- Verwendung UpdateImageSetMetadata mit einem AWS SDK oder CLI

Verwendung **`CopyImageSet`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CopyImageSet` verwendet wird.

CLI

AWS CLI

Beispiel 1: So kopieren Sie einen Bildsatz ohne Ziel.

Im folgenden Beispiel für `copy-image-set` wird eine doppelte Kopie eines Bildersatzes ohne Ziel erstellt.

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

Ausgabe:

```
{
    "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"
}
```

Beispiel 2: So kopieren Sie einen Bildersatz mit Ziel.

Im folgenden Beispiel für `copy-image-set` wird eine doppelte Kopie eines Bildersatzes mit Ziel erstellt.

```
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"} }'
```

Ausgabe:

```
{
```

```
    "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"
}
```

Beispiel 3: So kopieren Sie eine Teilmenge von Instances aus einem Quellbildersatz in einen Zielbildersatz.

Im folgenden Beispiel für `copy-image-set` wird eine DICOM-Instance aus dem Quellbildersatz in den Zielbildersatz kopiert. Der force-Parameter dient dazu, Inkonsistenzen in den Attributen auf Patienten-, Studien- und Serienebene zu überschreiben.

```
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
```

Ausgabe:

```
{
```

```
    "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"
}
```

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Kopieren eines Bildsatzes.

- Einzelheiten zur API finden Sie CopyImageSetin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
        }
    }
```

Hilfsfunktion zum Erstellen kopierbarer Attribute.

```
    /**
     * 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();
    }
```

- Einzelheiten zur API finden Sie CopyImageSetin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

Hilfsfunktion zum Kopieren eines Imagesatzes.

```
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 = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
  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: 'xxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
    //             latestVersionId: '4',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //      }
    // }
    return response;
  } catch (err) {
    console.error(err);
```

```
  }
};
```

Kopieren Sie einen Imagesatz ohne Ziel.

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

Kopieren Sie einen Imagesatz mit Ziel.

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

Kopieren Sie eine Teilmenge eines Imagesatzes mit einem Ziel und erzwingen Sie den Kopiervorgang.

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

- Einzelheiten zur API finden Sie CopyImageSetin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

Hilfsfunktion zum Kopieren eines Imagesatzes.

```
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"]
```

Kopieren Sie einen Imagesatz ohne Ziel.

```
            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,
            )
```

Kopieren Sie einen Imagesatz mit Ziel.

```
            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,
            )
```

Kopieren Sie eine Teilmenge eines Imagesatzes.

```
            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,
            )
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie CopyImageSetin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_image_set_id = '12345678901234567890123456789012345678901234567890'
        " 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.
```

- Einzelheiten zur API finden Sie CopyImageSetin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`CreateDatastore`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CreateDatastore` verwendet wird.

Bash

AWS CLI mit Bash-Skript

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

###############################################################################
# function imaging_create_datastore
#
# This function creates an AWS HealthImaging data store for importing DICOM P10
 files.
#
```

```
# Parameters:
#       -n data_store_name - The name of the data store.
#
# Returns:
#       The datastore ID.
#    And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function imaging_create_datastore() {
  local datastore_name response
  local option OPTARG # Required to use getopts command in a function.

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

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

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

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

  local error_code=${?}

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

  echo "$response"

  return 0
}
```

- Einzelheiten zur API finden Sie CreateDatastorein der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: Um einen Datenspeicher zu erstellen

Im folgenden Codebeispiel für `create-datastore` wird ein Datenspeicher mit dem Namen `my-datastore` erstellt. Wenn Sie einen Datenspeicher ohne Angabe von a erstellen`--lossless-storage-format`, wird AWS HealthImaging standardmäßig HTJ2 K (High Throughput JPEG 2000) verwendet.

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

Ausgabe:

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

Beispiel 2: So erstellen Sie einen Datenspeicher mit dem verlustfreien JPEG 2000-Speicherformat

Ein Datenspeicher, der mit dem verlustfreien Speicherformat JPEG 2000 konfiguriert ist, transkodiert und speichert verlustfreie Bildframes im JPEG 2000-Format. Bildframes können dann ohne Transcodierung im Format JPEG 2000 Lossless abgerufen werden. Im folgenden `create-datastore` Codebeispiel wird ein für das verlustfreies JPEG 2000-Speicherformat konfigurierter Datenspeicher mit dem Namen `my-datastore` erstellt.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Erstellen eines Datenspeichers.

- Einzelheiten zur API finden Sie CreateDatastoreunter AWS CLI Befehlsreferenz.

Java

SDK für 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 "";
    }
```

- Einzelheiten zur API finden Sie CreateDatastorein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
```

```
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie CreateDatastorein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie CreateDatastorein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie CreateDatastorein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteDatastore`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteDatastore` verwendet wird.

Bash

AWS CLI mit Bash-Skript

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

##############################################################################
# function imaging_delete_datastore
```

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

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

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

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

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

  local error_code=${?}

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

  return 0
}
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So löschen Sie einen Datenspeicher

Im folgenden Codebeispiel für `delete-datastore` wird ein Datenspeicher gelöscht.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Löschen eines Datenspeichers.

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
       }
   }
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store to delete.
```

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

  return response;
};
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie DeleteDatastorein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    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.
```

- Einzelheiten zur API finden Sie DeleteDatastorein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteImageSet`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteImageSet` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Erste Schritte mit Bildsätzen und Bild-Frames

C++

SDK für 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();
}
```

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So löschen Sie einen Bildersatz

Im folgenden Codebeispiel für `delete-image-set` wird ein Bildersatz gelöscht.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Löschen eines Bilddatensatzes.

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS CLI Befehlsreferenz.

Java

SDK für 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);
        }
    }
```

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
```

```
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie DeleteImageSetin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie DeleteImageSetin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`GetDICOMImportJob`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetDICOMImportJob` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Erste Schritte mit Bildsätzen und Bild-Frames

C++

SDK für 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;
}
```

- Einzelheiten zur API finden Sie unter Get DICOMImport Job in der AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So rufen Sie die Eigenschaften eines DICOM-Importauftrags ab

Im folgenden Codebeispiel für `get-dicom-import-job` werden die Eigenschaften eines DICOM-Importauftrags abgerufen.

```
aws medical-imaging get-dicom-import-job \
    --datastore-id "12345678901234567890123456789012" \
    --job-id "09876543210987654321098765432109"
```

Ausgabe:

```
{
    "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/"
    }
}
```

Weitere Informationen finden Sie im AWS HealthImaging Developer Guide unter Abrufen der Eigenschaften von Importaufträgen.

- API-Einzelheiten finden Sie unter Get DICOMImport Job in der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie unter Get DICOMImport Job in der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

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

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

  return response;
};
```

- Einzelheiten zur API finden Sie unter Get DICOMImport Job in der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- API-Einzelheiten finden Sie unter Get DICOMImport Job in AWS SDK for Python (Boto3) API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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-Details finden Sie unter Get DICOMImport Job in AWS SDK for SAP ABAP API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`GetDatastore`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetDatastore` verwendet wird.

Bash

AWS CLI mit Bash-Skript

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

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

```
    echo "Gets a data store's properties."
    echo "  -i datastore_id - The ID of the data store."
    echo ""
  }

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

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

  local response

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

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

```
    return 1
  fi

  echo "$response"

  return 0
}
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: Um die Eigenschaften eines Datenspeichers abzurufen

Im folgenden Codebeispiel für `get-datastore` werden die Eigenschaften eines Datenspeichers abgerufen.

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

Ausgabe:

```
{
    "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"
    }
```

```
}
```

Beispiel 2: Um die Eigenschaften eines Datenspeichers zu konfigurieren für JPEG2000

Im folgenden `get-datastore` Codebeispiel werden die Eigenschaften eines Datenspeichers für einen Datenspeicher abgerufen, der für das Format JPEG 2000 Lossless konfiguriert ist.

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

Ausgabe:

```
{
    "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"
    }
}
```

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Datenspeichereigenschaften.

- Einzelheiten zur API finden Sie GetDatastorein der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetDatastorein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    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.
```

- Einzelheiten zur API finden Sie GetDatastorein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`GetImageFrame`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetImageFrame` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Erste Schritte mit Bildsätzen und Bild-Frames

C++

SDK für 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();
}
```

- Einzelheiten zur API finden Sie GetImageFramein der AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So rufen Sie Pixeldaten des Bildersatzes ab

Im folgenden Codebeispiel für `get-image-frame` wird ein Image-Frame abgerufen.

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

Hinweis: Dieses Codebeispiel beinhaltet keine Ausgabe, da die GetImageFrame Aktion einen Stream von Pixeldaten an die Datei imageframe.jph zurückgibt. Informationen zum Dekodieren und Anzeigen von Bildrahmen finden Sie unter K-Decodierungsbibliotheken. HTJ2

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Pixeldaten von Bilddatensätzen.

- Einzelheiten zur API finden Sie GetImageFramein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
                }
        }
```

- Einzelheiten zur API finden Sie GetImageFramein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie GetImageFramein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetImageFramein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_frame_id = '12345678901234567890123456789012345678901234567890'
        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.
```

- Einzelheiten zur API finden Sie GetImageFramein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`GetImageSet`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetImageSet` verwendet wird.

CLI

AWS CLI

So rufen Sie Bildsatzeigenschaften ab

Im folgenden Codebeispiel für `get-image-set` werden die Eigenschaften für einen Bildsatz abgerufen.

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

Ausgabe:

```
{
```

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Bilddatensatz-Eigenschaften.

- Einzelheiten zur API finden Sie GetImageSetin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie GetImageSetin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
```

```
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie GetImageSetin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetImageSetin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " 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.
```

- Einzelheiten zur API finden Sie GetImageSetin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`GetImageSetMetadata`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetImageSetMetadata` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Erste Schritte mit Bildsätzen und Bild-Frames

C++

SDK für C++

Hilfsfunktion zum Abrufen von Bildsatz-Metadaten.

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

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

    return outcome.IsSuccess();
}
```

Rufen Sie Bildsatz-Metadaten ohne Version ab.

```
        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;
        }
```

Rufen Sie Bildsatz-Metadaten mit Version ab.

```
        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;
        }
```

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: So rufen Sie Bildsatz-Metadaten ohne Version ab

Im folgenden Codebeispiel für `get-image-set-metadata` werden Metadaten für einen Bildsatz abgerufen, ohne eine Version anzugeben.

Anmerkung: Der Parameter `outfile` ist erforderlich.

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

Die zurückgegebenen Metadaten werden mit GZIP komprimiert und in der Datei „studymetadata.json.gz“ gespeichert. Um den Inhalt des zurückgegebenen JSON-Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren.

Ausgabe:

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

Beispiel 2: So rufen Sie Bildsatz-Metadaten mit Version ab

Im folgenden Codebeispiel für `get-image-set-metadata` werden Metadaten für einen Bildsatz mit einer bestimmten Version abgerufen.

Anmerkung: Der Parameter `outfile` ist erforderlich.

```
aws medical-imaging get-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
```

```
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --version-id 1 \
    studymetadata.json.gz
```

Die zurückgegebenen Metadaten werden mit GZIP komprimiert und in der Datei „studymetadata.json.gz“ gespeichert. Um den Inhalt des zurückgegebenen JSON-Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren.

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Abrufen von Bilddatensatz-Metadaten.

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS CLI Befehlsreferenz.

Java

SDK für 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);
        }
    }
```

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

Hilfsfunktion zum Abrufen von Bildsatz-Metadaten.

```
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 = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
```

```
  versionID = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

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

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

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

  return response;
};
```

Rufen Sie Bildsatz-Metadaten ohne Version ab.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
    );
  } catch (err) {
```

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

Rufen Sie Bildsatz-Metadaten mit Version ab.

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

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

Hilfsfunktion zum Abrufen von Bildsatz-Metadaten.

```
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
```

Rufen Sie Bildsatz-Metadaten ohne Version ab.

```
                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
```

```
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
```

Rufen Sie Bildsatz-Metadaten mit Version ab.

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

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie GetImageSetMetadatain AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie GetImageSetMetadatain der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ListDICOMImportJobs`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListDICOMImportJobs` verwendet wird.

CLI

AWS CLI

So listen Sie DICOM-Importaufträge auf

Im folgenden Codebeispiel für `list-dicom-import-jobs` werden die DICOM-Importaufträge aufgelistet.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Developer Guide unter Auflisten von Importaufträgen.

- Einzelheiten zur API finden Sie unter DICOMImportJobs auflisten in der AWS CLI Befehlsreferenz.

Java

SDK für 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<>();
    }
```

- Einzelheiten zur API finden Sie unter DICOMImportJobs in der AWS SDK for Java 2.x API-Referenz auflisten.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

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

```
    pageSize: 50,
  };

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

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

  return jobSummaries;
};
```

- Einzelheiten zur API finden Sie unter DICOMImportJobs auflisten in der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- API-Einzelheiten finden Sie unter API-Referenz zum DICOMImportAuflisten von Aufträgen in AWS SDK for Python (Boto3).

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        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.
```

- Einzelheiten zur API finden Sie in der Referenz zum DICOMImportAuflisten von Aufträgen im AWS SDK für SAP ABAP API.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ListDatastores`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListDatastores` verwendet wird.

Bash

AWS CLI mit Bash-Skript

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

###############################################################################
# function imaging_list_datastores
#
# List the HealthImaging data stores in the account.
#
# Returns:
#       [[datastore_name, datastore_id, datastore_status]]
#    And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function imaging_list_datastores() {
```

```
  local option OPTARG # Required to use getopts command in a function.
  local error_code
  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_list_datastores"
    echo "Lists the AWS HealthImaging data stores in the account."
    echo ""
  }

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

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

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

  echo "$response"

  return 0
}
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS CLI Befehlsreferenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So listen Sie Datenspeicher auf

Im folgenden Codebeispiel für `list-datastores` werden die verfügbaren Datenspeicher aufgelistet.

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

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Developer Guide unter Auflisten von Datenspeichern.

- Einzelheiten zur API finden Sie ListDatastoresin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
   }
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

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

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

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

  return datastoreSummaries;
};
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie ListDatastoresin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie ListDatastoresin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ListImageSetVersions`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListImageSetVersions` verwendet wird.

CLI

AWS CLI

So listen Sie Versionen von Bildsätzen auf

Im folgendem Codebeispiel für `list-image-set-versions` wird der Versionsverlauf für einen Bildsatz aufgelistet.

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

Ausgabe:

```
{
    "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
        }
    ]
}
```

Weitere Informationen finden Sie im AWS HealthImaging Developer Guide unter Auflisten von Imageset-Versionen.

- Einzelheiten zur API finden Sie ListImageSetVersionsin der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie ListImageSetVersionsin der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Einzelheiten zur API finden Sie ListImageSetVersionsin der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie ListImageSetVersionsin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie ListImageSetVersionsin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ListTagsForResource`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListTagsForResource` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Taggen eines Datenspeichers
- Taggen eines Imagesatzes

CLI

AWS CLI

Beispiel 1: So listen Sie Ressourcen-Tags für einen Datenspeicher auf

Im folgenden Codebeispiel für `list-tags-for-resource` werden die Tags für einen Datenspeicher aufgelistet.

```
aws medical-imaging list-tags-for-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"
```

Ausgabe:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

Beispiel 2: So listen Sie Ressourcen-Tags für einen Bildsatz auf

Im folgenden Codebeispiel für `list-tags-for-resource` werden die Tags für einen Bildsatz aufgelistet.

```
aws medical-imaging list-tags-for-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b"
```

Ausgabe:

```
{
    "tags":{
        "Deployment":"Development"
    }
}
```

Weitere Informationen finden Sie unter Ressourcen taggen mit AWS HealthImaging im AWS HealthImaging Entwicklerhandbuch.

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS CLI Befehlsreferenz.

Java

SDK für 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;
    }
```

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie ListTagsForResourcein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie ListTagsForResourcein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`SearchImageSets`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `SearchImageSets` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Erste Schritte mit Bildsätzen und Bild-Frames

C++

SDK für C++

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
//! 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;
}
```

Anwendungsfall 1: EQUAL-Operator.

```
       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;
           }
       }
```

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

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

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

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

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

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

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

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

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

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

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
        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;
            }
        }
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
        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;
            }
        }
```

- Einzelheiten zur API finden Sie in der API-Referenz. SearchImageSetsAWS SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

Beispiel 1: So suchen Sie nach Bilddatensätzen mit einem EQUAL-Operator

Im folgenden Codebeispiel für `search-image-sets` wird der EQUAL-Operator verwendet, um Bildsätze auf der Grundlage eines bestimmten Werts zu durchsuchen.

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

Inhalt von `search-criteria.json`

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

Ausgabe:

```
{
    "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"
```

```
    }]
}
```

Beispiel 2: Um Bilddatensätze mit einem BETWEEN-Operator unter Verwendung von DICOMStudy Datum und DICOMStudy Uhrzeit zu durchsuchen

Im folgenden Codebeispiel für `search-image-sets` wird nach Bildsätzen mit DICOM-Studien gesucht, die zwischen dem 1. Januar 1990 (12:00 Uhr) und dem 1. Januar 2023 (12:00 Uhr) generiert wurden.

Hinweis: Die DICOMStudy Uhrzeit ist optional. Wenn der Parameter nicht vorhanden ist, ist 12:00 Uhr (Beginn des Tages) der Zeitwert für die Datumsangaben, die für die Filterung bereitgestellt werden.

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

Inhalt von `search-criteria.json`

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

Ausgabe:

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

Beispiel 3: So suchen Sie Bildsätze mit einem BETWEEN-Operator mithilfe von createdAt (Zeitstudien wurden bisher fortgeführt)

Im folgenden `search-image-sets` Codebeispiel wird nach Bilddatensätzen gesucht, bei denen DICOM-Studien HealthImaging zwischen den Zeitbereichen in der UTC-Zeitzone persistiert wurden.

Hinweis: Geben Sie „createdAt“ in dem im Beispiel aufgeführten Format an („1985-04-12T23:20:50.52Z“).

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

Inhalt von `search-criteria.json`

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

```
            "createdAt": "2022-04-12T23:20:50.52Z"
        }],
        "operator": "BETWEEN"
    }]
}
```

Ausgabe:

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

Beispiel 4: Um Bilddatensätze mit einem EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt zu durchsuchen und die Antwort in ASC-Reihenfolge im Feld updatedAt zu sortieren

Das folgende `search-image-sets` Codebeispiel sucht nach Bilddatensätzen mit einem EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiert die Antwort in ASC-Reihenfolge im Feld updatedAt.

Hinweis: Geben Sie „updatedAt“ in dem im Beispiel aufgeführten Format an („1985-04-12T23:20:50.52Z“).

```
aws medical-imaging search-image-sets \
```

```
    --datastore-id 12345678901234567890123456789012 \
    --search-criteria  file://search-criteria.json
```

Inhalt von `search-criteria.json`

```
{
    "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"
    }
}
```

Ausgabe:

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

Weitere Informationen finden Sie unter Suchen von Bilddatensätzen im Entwicklerhandbuch.AWS HealthImaging

- Einzelheiten zur API finden Sie SearchImageSetsin der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
   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;
   }
```

Anwendungsfall 1: EQUAL-Operator.

```
        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();
        }
```

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

```
        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();
        }
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
        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();
        }
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
        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();
        }
```

- Einzelheiten zur API finden Sie in der API-Referenz. SearchImageSetsAWS SDK for Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
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;
};
```

Anwendungsfall 1: EQUAL-Operator.

```
  const datastoreId = "12345678901234567890123456789012";

  try {
    const searchCriteria = {
      filters: [
        {
```

```
        values: [{ DICOMPatientId: "1234567" }],
        operator: "EQUAL",
      },
    ],
  };

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

Anwendungsfall #2: BETWEEN-Operator, der DICOMStudy Datum und DICOMStudy Uhrzeit verwendet.

```
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);
```

```
}
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
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);
}
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
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);
  }
```

- Einzelheiten zur API finden Sie in der API-Referenz. SearchImageSetsAWS SDK für JavaScript

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
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
```

Anwendungsfall 1: EQUAL-Operator.

```
        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}")
```

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

```
        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}"
        )
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
        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}"
        )
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das Feld updatedAt.

```
        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}")
```

MedicalImagingWrapper Der folgende Code instanziiert das Objekt.

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

- Einzelheiten zur API finden Sie SearchImageSetsin AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie SearchImageSetsin der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`StartDICOMImportJob`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `StartDICOMImportJob` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Erste Schritte mit Bildsätzen und Bild-Frames

C++

SDK für 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();
}
```

- Einzelheiten zur API finden Sie unter DICOMImportJob starten in der AWS SDK für C++ API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

CLI

AWS CLI

So starten Sie einen DICOM-Importauftrag

Im folgenden Codebeispiel für `start-dicom-import-job` wird ein DICOM-Importauftrage gestartet.

```
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"
```

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Einen Importauftrag starten.

- API-Einzelheiten finden Sie unter DICOMImportJob starten in der AWS CLI Befehlsreferenz.

Java

SDK für 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 "";
   }
```

- Einzelheiten zur API finden Sie unter DICOMImportJob starten in der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

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

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

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

- Einzelheiten zur API finden Sie unter DICOMImportJob starten in der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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"]
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie unter Start DICOMImport Job in AWS SDK for Python (Boto3) API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für SAP ABAP

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
```

```
        " 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-Details finden Sie unter Referenz zum Starten DICOMImport eines Job im AWS SDK für SAP ABAP API.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`TagResource`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `TagResource` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Taggen eines Datenspeichers
- Taggen eines Imagesatzes

CLI

AWS CLI

Beispiel 1: So markieren Sie einen Datenspeicher

In den folgenden Codebeispielen für `tag-resource` wird ein Datenspeicher mit Tags versehen.

```
aws medical-imaging tag-resource \
  --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
  --tags '{"Deployment":"Development"}'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Beispiel 2: So taggen Sie einen Bildsatz

In den folgenden Codebeispielen für `tag-resource` wird ein Bildsatz mit Tags versehen.

```
aws medical-imaging tag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
    --tags '{"Deployment":"Development"}'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Ressourcen taggen mit AWS HealthImaging im AWS HealthImaging Entwicklerhandbuch.

- Einzelheiten zur API finden Sie TagResourcein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
       }
   }
```

- Einzelheiten zur API finden Sie TagResourcein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie TagResourcein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie TagResourcein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie TagResourcein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`UntagResource`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `UntagResource` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Taggen eines Datenspeichers
- Taggen eines Imagesatzes

CLI

AWS CLI

Beispiel 1: So heben Sie die Markierung eines Datenspeichers auf

Im folgenden Codebeispiel für `untag-resource` werden Tags aus einem Datenspeicher entfernt.

```
aws medical-imaging untag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012" \
    --tag-keys '["Deployment"]'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Beispiel 2: So heben Sie das Tagging eines Bildsatzes auf

Im folgenden Codebeispiel für `untag-resource` werden Tags aus einem Bildsatz entfernt.

```
aws medical-imaging untag-resource \
    --resource-arn "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
    --tag-keys '["Deployment"]'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Ressourcen taggen mit AWS HealthImaging im AWS HealthImaging Entwicklerhandbuch.

- Einzelheiten zur API finden Sie UntagResourcein der AWS CLI Befehlsreferenz.

Java

SDK für 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);
        }
    }
```

- Einzelheiten zur API finden Sie UntagResourcein der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für 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;
};
```

- Einzelheiten zur API finden Sie UntagResourcein der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Einzelheiten zur API finden Sie UntagResourcein AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie UntagResourcein der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`UpdateImageSetMetadata`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `UpdateImageSetMetadata` verwendet wird.

CLI

AWS CLI

Beispiel 1: So fügen Sie ein Attribut in Bildsatz-Metadaten ein oder aktualisieren es

Im folgenden Beispiel für `update-image-set-metadata` wird ein Attribut in die Metadaten eines Bildsatzes eingefügt.

```
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
```

Inhalt von `metadata-updates.json`

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

Ausgabe:

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

Beispiel 2: So entfernen Sie ein Attribut aus den Metadaten eines Bildsatzes

Im folgenden Beispiel für `update-image-set-metadata` wird ein Attribut aus den Metadaten eines Bildsatzes entfernt.

```
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
```

Inhalt von `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "removableAttributes": "{\"SchemaVersion\":1.1,\"Study\":{\"DICOM\":
{\"StudyDescription\":\"CHEST\"}}}"
```

```
    }
}
```

Ausgabe:

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

Beispiel 3: So entfernen Sie eine Instance aus den Metadaten eines Bildsatzes

Im folgenden Beispiel für `update-image-set-metadata` wird eine Instance aus den Metadaten eines Bildsatzes entfernt.

```
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
```

Inhalt von `metadata-updates.json`

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

Ausgabe:

```
{
```

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

Beispiel 4: So setzen Sie einen Bildsatz auf eine frühere Version zurück

Das folgende `update-image-set-metadata` Beispiel zeigt, wie ein Bildsatz auf eine frühere Version zurückgesetzt wird. CopyImageSet und UpdateImageSetMetadata Aktionen erstellen neue Versionen von Bilddatensätzen.

```
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"}'
```

Ausgabe:

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

Beispiel 5: So fügen Sie einer Instance ein privates DICOM-Datenelement hinzu

Das folgende Beispiel für `update-image-set-metadata` zeigt, wie ein privates Element zu einer angegebenen Instance innerhalb eines Bildsatzes hinzugefügt wird. Der DICOM-Standard erlaubt private Datenelemente für die Übertragung von Informationen, die nicht in Standarddatenelementen enthalten sein dürfen. Mit der UpdateImageSetMetadata Aktion können Sie private Datenelemente erstellen, aktualisieren und löschen.

```
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
```

Inhalt von `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"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\": {\"DICOM\":
 {\"001910F9\": \"97\"},\"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}"
    }
}
```

Ausgabe:

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

Beispiel 6: So aktualisieren Sie ein privates DICOM-Datenelement in einer Instance

Das folgende Beispiel für `update-image-set-metadata` zeigt, wie der Wert eines privates Datenelements, das zu einer Instance eines Bildsatzes gehört, aktualisiert wird.

```
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
```

Inhalt von `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"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\": {\"DICOM\":
 {\"00091001\": \"GE_GENESIS_DD\"}}}}}}"
    }
}
```

Ausgabe:

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

Beispiel 7: Um eine SOPInstance UID mit dem Force-Parameter zu aktualisieren

Das folgende `update-image-set-metadata` Beispiel zeigt, wie eine SOPInstance UID aktualisiert wird, indem der Force-Parameter verwendet wird, um die DICOM-Metadatenbeschränkungen zu überschreiben.

```
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
```

Inhalt von `metadata-updates.json`

```
{
```

```
    "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\"}}}}}}}"
    }
}
```

Ausgabe:

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

Weitere Informationen finden Sie im AWS HealthImaging Entwicklerhandbuch unter Aktualisieren von Bilddatensatz-Metadaten.

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain der AWS CLI Befehlsreferenz.

Java

SDK für 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;
       }
   }
```

Anwendungsfall 1: Fügen Sie ein Attribut ein oder aktualisieren Sie es.

```
               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);
```

Anwendungsfall 2: Entfernen Sie ein Attribut.

```
                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);
```

Anwendungsfall 3: Entfernen Sie eine Instance.

```
                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);
```

Anwendungsfall 4: Kehren Sie zu einer früheren Version zurück.

```
                // 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);
```

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain der AWS SDK for Java 2.x API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

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

Anwendungsfall 1: Fügen Sie ein Attribut ein oder aktualisieren Sie es und erzwingen Sie das Update.

```
    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,
);
```

Anwendungsfall 2: Entfernen Sie ein Attribut.

```
// 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,
);
```

Anwendungsfall 3: Entfernen Sie eine Instance.

```
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,
  );
```

Anwendungsfall 4: Kehren Sie zu einer früheren Version zurück.

```
  const updateMetadata = {
    revertToVersionId: "1",
  };

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

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain der AWS SDK für JavaScript API-Referenz.

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für 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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

Anwendungsfall 1: Fügen Sie ein Attribut ein oder aktualisieren Sie es.

```
            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
            )
```

Anwendungsfall 2: Entfernen Sie ein Attribut.

```
            # 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
            )
```

Anwendungsfall 3: Entfernen Sie eine Instance.

```
            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
            )
```

Anwendungsfall 4: Kehren Sie zu einer früheren Version zurück.

```
            metadata = {"revertToVersionId": "1"}

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

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

SAP ABAP

SDK für 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.
```

- Einzelheiten zur API finden Sie UpdateImageSetMetadatain der AWS API-Referenz zum SDK für SAP ABAP.

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Szenarien für die HealthImaging Verwendung AWS SDKs

Die folgenden Codebeispiele zeigen Ihnen, wie Sie gängige Szenarien in HealthImaging with implementieren AWS SDKs. Diese Szenarien zeigen Ihnen, wie Sie bestimmte Aufgaben erledigen können, indem Sie mehrere Funktionen innerhalb HealthImaging oder in Kombination mit anderen aufrufen AWS-Services. Jedes Szenario enthält einen Link zum vollständigen Quell-Code, wo Sie Anleitungen zum Einrichten und Ausführen des Codes finden.

Szenarien zielen auf eine mittlere Erfahrungsebene ab, um Ihnen zu helfen, Service-Aktionen im Kontext zu verstehen.

Beispiele

- Erste Schritte mit HealthImaging Bildsätzen und Bildrahmen mithilfe eines AWS SDK
- Kennzeichnen eines HealthImaging Datenspeichers mithilfe eines SDK AWS
- Taggen eines HealthImaging Bilddatensatzes mithilfe eines SDK AWS

Erste Schritte mit HealthImaging Bildsätzen und Bildrahmen mithilfe eines AWS SDK

Die folgenden Codebeispiele zeigen, wie Sie DICOM-Dateien importieren und Bildrahmen herunterladen. HealthImaging

Die Implementierung ist als Befehlszeilenanwendung strukturiert.

- Richten Sie Ressourcen für einen DICOM-Import ein.
- Importieren Sie DICOM-Dateien in einen Datenspeicher.
- Rufen Sie den Bildsatz IDs für den Importjob ab.
- Rufen Sie den Bildrahmen IDs für die Bildsätze ab.
- Laden Sie die Bild-Frames herunter, dekodieren Sie sie und überprüfen Sie sie.
- Bereinigen Sie die Ressourcen.

C++

SDK für C++

Erstellen Sie einen CloudFormation Stack mit den erforderlichen Ressourcen.

```
   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: ");
    }
```

Kopieren Sie DICOM-Dateien in den Amazon-S3-Import-Bucket.

```
    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;
   }
```

Importieren Sie die DICOM-Dateien in den Amazon-S3-Datenspeicher.

```
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;
}
```

Rufen Sie Bildsätze ab, die durch den DICOM-Importauftrag erstellt wurden.

```
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;
}
```

Rufen Sie Bild-Frame-Informationen für Bildsätze ab.

```
bool AwsDoc::Medical_Imaging::getImageFramesForImageSet(const Aws::String
 &dataStoreID,
                                                        const Aws::String
 &imageSetID,
                                                        const Aws::String
 &outDirectory,

 Aws::Vector<ImageFrameInfo> &imageFrames,
                                                        const
 Aws::Client::ClientConfiguration &clientConfiguration) {
```

```
   Aws::String fileName = outDirectory + "/" + imageSetID +
"_metadata.json.gzip";
   bool result = false;
   if (getImageSetMetadata(dataStoreID, imageSetID, "", // Empty string for
version ID.
                           fileName, clientConfiguration)) {
       try {
           std::string metadataGZip;
           {
               std::ifstream inFileStream(fileName.c_str(), std::ios::binary);
               if (!inFileStream) {
                   throw std::runtime_error("Failed to open file " + fileName);
               }

               std::stringstream stringStream;
               stringStream << inFileStream.rdbuf();
               metadataGZip = stringStream.str();
           }
           std::string metadataJson = gzip::decompress(metadataGZip.data(),
                                                       metadataGZip.size());
           // Use JMESPath to extract the image set IDs.
           // https://jmespath.org/specification.html
           jsoncons::json doc = jsoncons::json::parse(metadataJson);
           std::string jmesPathExpression = "Study.Series.*.Instances[].*[]";
           jsoncons::json instances = jsoncons::jmespath::search(doc,

jmesPathExpression);
           for (auto &instance: instances.array_range()) {
               jmesPathExpression = "DICOM.RescaleSlope";
               std::string rescaleSlope = jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();
               jmesPathExpression = "DICOM.RescaleIntercept";
               std::string rescaleIntercept =
jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();

               jmesPathExpression = "ImageFrames[][]";
               jsoncons::json imageFramesJson =
jsoncons::jmespath::search(instance,

jmesPathExpression);
```

```
                for (auto &imageFrame: imageFramesJson.array_range()) {
                    ImageFrameInfo imageFrameIDs;
                    imageFrameIDs.mImageSetId = imageSetID;
                    imageFrameIDs.mImageFrameId = imageFrame.find(
                            "ID")->value().as_string();
                    imageFrameIDs.mRescaleIntercept = rescaleIntercept;
                    imageFrameIDs.mRescaleSlope = rescaleSlope;
                    imageFrameIDs.MinPixelValue = imageFrame.find(
                            "MinPixelValue")->value().as_string();
                    imageFrameIDs.MaxPixelValue = imageFrame.find(
                            "MaxPixelValue")->value().as_string();

                    jmesPathExpression =
 "max_by(PixelDataChecksumFromBaseToFullResolution, &Width).Checksum";
                    jsoncons::json checksumJson =
 jsoncons::jmespath::search(imageFrame,

 jmesPathExpression);
                    imageFrameIDs.mFullResolutionChecksum =
 checksumJson.as_integer<uint32_t>();

                    imageFrames.emplace_back(imageFrameIDs);
                }
            }

            result = true;
        }
        catch (const std::exception &e) {
            std::cerr << "getImageFramesForImageSet failed because " << e.what()
                      << std::endl;
        }
    }

    return result;
}

//! Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The HealthImaging image set ID.
  \param versionID: The HealthImaging image set version ID, ignored if empty.
  \param outputFilePath: The path where the metadata will be stored as gzipped
 json.
  \param clientConfig: Aws client configuration.
```

```
  \\return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
 &outputFilePath,
                                                  const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    if (!versionID.empty()) {
        request.SetVersionId(versionID);
    }
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
 client.GetImageSetMetadata(
            request);
    if (outcome.IsSuccess()) {
        std::ofstream file(outputFilePath, std::ios::binary);
        auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
        file << metadata.rdbuf();
    }
    else {
        std::cerr << "Failed to get image set metadata: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

Laden Sie Bild-Frames herunter, dekodieren Sie sie und überprüfen Sie sie.

```
bool AwsDoc::Medical_Imaging::downloadDecodeAndCheckImageFrames(
        const Aws::String &dataStoreID,
        const Aws::Vector<ImageFrameInfo> &imageFrames,
        const Aws::String &outDirectory,
        const Aws::Client::ClientConfiguration &clientConfiguration) {

    Aws::Client::ClientConfiguration clientConfiguration1(clientConfiguration);
```

```
   clientConfiguration1.executor =
Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>(
           "executor", 25);
   Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(
           clientConfiguration1);

   Aws::Utils::Threading::Semaphore semaphore(0, 1);
   std::atomic<size_t> count(imageFrames.size());

   bool result = true;
   for (auto &imageFrame: imageFrames) {
       Aws::MedicalImaging::Model::GetImageFrameRequest getImageFrameRequest;
       getImageFrameRequest.SetDatastoreId(dataStoreID);
       getImageFrameRequest.SetImageSetId(imageFrame.mImageSetId);

       Aws::MedicalImaging::Model::ImageFrameInformation imageFrameInformation;
       imageFrameInformation.SetImageFrameId(imageFrame.mImageFrameId);
       getImageFrameRequest.SetImageFrameInformation(imageFrameInformation);

       auto getImageFrameAsyncLambda = [&semaphore, &result, &count, imageFrame,
outDirectory](
               const Aws::MedicalImaging::MedicalImagingClient *client,
               const Aws::MedicalImaging::Model::GetImageFrameRequest &request,
               Aws::MedicalImaging::Model::GetImageFrameOutcome outcome,
               const std::shared_ptr<const Aws::Client::AsyncCallerContext>
&context) {

               if (!handleGetImageFrameResult(outcome, outDirectory,
imageFrame)) {
                   std::cerr << "Failed to download and convert image frame: "
                             << imageFrame.mImageFrameId << " from image set: "
                             << imageFrame.mImageSetId << std::endl;
                   result = false;
               }

               count--;
               if (count <= 0) {

                   semaphore.ReleaseAll();
               }
       }; // End of 'getImageFrameAsyncLambda' lambda.

       medicalImagingClient.GetImageFrameAsync(getImageFrameRequest,
                                               getImageFrameAsyncLambda);
```

```
    }

    if (count > 0) {
        semaphore.WaitOne();
    }

    if (result) {
        std::cout << imageFrames.size() << " image files were downloaded."
                  << std::endl;
    }

    return result;
}

bool AwsDoc::Medical_Imaging::decodeJPHFileAndValidateWithChecksum(
        const Aws::String &jphFile,
        uint32_t crc32Checksum) {
    opj_image_t *outputImage = jphImageToOpjBitmap(jphFile);
    if (!outputImage) {
        return false;
    }

    bool result = true;
    if (!verifyChecksumForImage(outputImage, crc32Checksum)) {
        std::cerr << "The checksum for the image does not match the expected
 value."
                  << std::endl;
        std::cerr << "File :" << jphFile << std::endl;
        result = false;
    }

    opj_image_destroy(outputImage);

    return result;
}

opj_image *
AwsDoc::Medical_Imaging::jphImageToOpjBitmap(const Aws::String &jphFile) {
    opj_stream_t *inFileStream = nullptr;
    opj_codec_t *decompressorCodec = nullptr;
    opj_image_t *outputImage = nullptr;
    try {
        std::shared_ptr<opj_dparameters> decodeParameters =
 std::make_shared<opj_dparameters>();
```

```
        memset(decodeParameters.get(), 0, sizeof(opj_dparameters));

        opj_set_default_decoder_parameters(decodeParameters.get());

        decodeParameters->decod_format = 1; // JP2 image format.
        decodeParameters->cod_format = 2; // BMP image format.

        std::strncpy(decodeParameters->infile, jphFile.c_str(),
                     OPJ_PATH_LEN);

        inFileStream = opj_stream_create_default_file_stream(
                decodeParameters->infile, true);
        if (!inFileStream) {
            throw std::runtime_error(
                    "Unable to create input file stream for file '" + jphFile +
"'.");
        }

        decompressorCodec = opj_create_decompress(OPJ_CODEC_JP2);
        if (!decompressorCodec) {
            throw std::runtime_error("Failed to create decompression codec.");
        }

        int decodeMessageLevel = 1;
        if (!setupCodecLogging(decompressorCodec, &decodeMessageLevel)) {
            std::cerr << "Failed to setup codec logging." << std::endl;
        }

        if (!opj_setup_decoder(decompressorCodec, decodeParameters.get())) {
            throw std::runtime_error("Failed to setup decompression codec.");
        }
        if (!opj_codec_set_threads(decompressorCodec, 4)) {
            throw std::runtime_error("Failed to set decompression codec
threads.");
        }

        if (!opj_read_header(inFileStream, decompressorCodec, &outputImage)) {
            throw std::runtime_error("Failed to read header.");
        }

        if (!opj_decode(decompressorCodec, inFileStream,
                        outputImage)) {
            throw std::runtime_error("Failed to decode.");
        }
```

```
        if (DEBUGGING) {
            std::cout << "image width : " << outputImage->x1 - outputImage->x0
                      << std::endl;
            std::cout << "image height : " << outputImage->y1 - outputImage->y0
                      << std::endl;
            std::cout << "number of channels: " << outputImage->numcomps
                      << std::endl;
            std::cout << "colorspace : " << outputImage->color_space <<
 std::endl;
        }

    } catch (const std::exception &e) {
        std::cerr << e.what() << std::endl;
        if (outputImage) {
            opj_image_destroy(outputImage);
            outputImage = nullptr;
        }
    }
    if (inFileStream) {
        opj_stream_destroy(inFileStream);
    }
    if (decompressorCodec) {
        opj_destroy_codec(decompressorCodec);
    }

    return outputImage;
}

//! Template function which converts a planar image bitmap to an interleaved
 image bitmap and
//! then verifies the checksum of the bitmap.
/*!
 * @param image: The OpenJPEG image struct.
 * @param crc32Checksum: The CRC32 checksum.
 * @return  bool: Function succeeded.
 */
template<class myType>
bool verifyChecksumForImageForType(opj_image_t *image, uint32_t crc32Checksum) {
    uint32_t width = image->x1 - image->x0;
    uint32_t height = image->y1 - image->y0;
    uint32_t numOfChannels = image->numcomps;

    // Buffer for interleaved bitmap.
```

```
    std::vector<myType> buffer(width * height * numOfChannels);

    // Convert planar bitmap to interleaved bitmap.
    for (uint32_t channel = 0; channel < numOfChannels; channel++) {
        for (uint32_t row = 0; row < height; row++) {
            uint32_t fromRowStart = row / image->comps[channel].dy * width /
                                    image->comps[channel].dx;
            uint32_t toIndex = (row * width) * numOfChannels + channel;

            for (uint32_t col = 0; col < width; col++) {
                uint32_t fromIndex = fromRowStart + col / image-
>comps[channel].dx;

                buffer[toIndex] = static_cast<myType>(image-
>comps[channel].data[fromIndex]);

                toIndex += numOfChannels;
            }
        }
    }

    // Verify checksum.
    boost::crc_32_type crc32;
    crc32.process_bytes(reinterpret_cast<char *>(buffer.data()),
                        buffer.size() * sizeof(myType));

    bool result = crc32.checksum() == crc32Checksum;
    if (!result) {
        std::cerr << "verifyChecksumForImage, checksum mismatch, expected - "
                  << crc32Checksum << ", actual - " << crc32.checksum()
                  << std::endl;
    }

    return result;
}

//! Routine which verifies the checksum of an OpenJPEG image struct.
/*!
 * @param image: The OpenJPEG image struct.
 * @param crc32Checksum: The CRC32 checksum.
 * @return  bool: Function succeeded.
 */
bool AwsDoc::Medical_Imaging::verifyChecksumForImage(opj_image_t *image,
                                                     uint32_t crc32Checksum) {
```

```
    uint32_t channels = image->numcomps;
    bool result = false;
    if (0 < channels) {
        // Assume the precision is the same for all channels.
        uint32_t precision = image->comps[0].prec;
        bool signedData = image->comps[0].sgnd;
        uint32_t bytes = (precision + 7) / 8;

        if (signedData) {
            switch (bytes) {
                case 1 :
                    result = verifyChecksumForImageForType<int8_t>(image,

crc32Checksum);
                    break;
                case 2 :
                    result = verifyChecksumForImageForType<int16_t>(image,

crc32Checksum);
                    break;
                case 4 :
                    result = verifyChecksumForImageForType<int32_t>(image,

crc32Checksum);
                    break;
                default:
                    std::cerr
                            << "verifyChecksumForImage, unsupported data type,
signed bytes - "
                            << bytes << std::endl;
                    break;
            }
        }
        else {
            switch (bytes) {
                case 1 :
                    result = verifyChecksumForImageForType<uint8_t>(image,

crc32Checksum);
                    break;
                case 2 :
                    result = verifyChecksumForImageForType<uint16_t>(image,

crc32Checksum);
```

```
                    break;
                case 4 :
                    result = verifyChecksumForImageForType<uint32_t>(image,

 crc32Checksum);
                    break;
                default:
                    std::cerr
                            << "verifyChecksumForImage, unsupported data type,
 unsigned bytes - "
                            << bytes << std::endl;
                    break;
            }
        }

        if (!result) {
            std::cerr << "verifyChecksumForImage, error bytes " << bytes
                      << " signed "
                      << signedData << std::endl;
        }
    }
    else {
        std::cerr << "'verifyChecksumForImage', no channels in the image."
                  << std::endl;
    }
    return result;
}
```

Bereinigen von Ressourcen.

```
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;
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für C++ -API-Referenz.
  - DeleteImageSet
  - DICOMImportJob bekommen
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - DICOMImportJob starten

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

Orchestrieren Sie die Schritte (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]",
  });
}
```

Stellen Sie Ressourcen bereit (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 },
);
```

Kopieren Sie DICOM-Dateien (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.`,
);
```

Starten Sie den Import in den Datenspeicher (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}`,
);
```

Bilddatensatz abrufen IDs (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")}`,
);
```

Holen Sie sich den Bildrahmen IDs (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`;
    }

    return output;
  },
);
```

Überprüfen Sie die Image-Rahmen (verify-steps.js). Die Bibliothek zur Überprüfung von AWS HealthImaging Pixeldaten wurde zur Überprüfung verwendet.

```
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
```

```
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */
```

```
/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
    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 [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid}
 and sop ${sopInstanceUid}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
```

```
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(
                    `Verification tool exited with code ${code} for image set
 ${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

Vernichten Sie Ressourcen (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,
  },
);
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - DeleteImageSet
  - DICOMImportJob bekommen
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - DICOMImportJob starten

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

Erstellen Sie einen CloudFormation Stack mit den erforderlichen Ressourcen.

```
    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
```

Kopieren Sie DICOM-Dateien in den Amazon-S3-Import-Bucket.

```
   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.")
```

Importieren Sie die DICOM-Dateien in den Amazon-S3-Datenspeicher.

```
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"]
```

Rufen Sie Bildsätze ab, die durch den DICOM-Importauftrag erstellt wurden.

```
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
```

Rufen Sie Bild-Frame-Informationen für Bildsätze ab.

```
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
```

Laden Sie Bild-Frames herunter, dekodieren Sie sie und überprüfen Sie sie.

```
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\tImage checksum verified for {image_frame['imageFrameId']}:
 {image_result }"
            )
            total_result = total_result and image_result
        return total_result

    @staticmethod
```

```
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array
```

Bereinigen Sie die Ressourcen.

```
    def destroy(self, stack):
        """
        Destroys the resources managed by the CloudFormation stack, and the
CloudFormation
        stack itself.

        :param stack: The CloudFormation stack that manages the example
resources.
        """

        print(f"\t\tCleaning up resources and {stack.name}.")
        data_store_id = None
        for oput in stack.outputs:
            if oput["OutputKey"] == "DatastoreID":
                data_store_id = oput["OutputValue"]
        if data_store_id is not None:
            print(f"\t\tDeleting image sets in data store {data_store_id}.")
            image_sets = self.medical_imaging_wrapper.search_image_sets(
                data_store_id, {}
            )
            image_set_ids = [image_set["imageSetId"] for image_set in image_sets]

            for image_set_id in image_set_ids:
                self.medical_imaging_wrapper.delete_image_set(
                    data_store_id, image_set_id
```

```
                )
                print(f"\t\tDeleted image set with id : {image_set_id}")

        print(f"\t\tDeleting {stack.name}.")
        stack.delete()
        print("\t\tWaiting for stack removal. This may take a few minutes.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_delete_complete")
        waiter.wait(StackName=stack.name)
        print("\t\tStack delete complete.")




class MedicalImagingWrapper:
    """Encapsulates 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
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - DeleteImageSet
  - DICOMImportJob bekommen

- GetImageFrame
- GetImageSetMetadata
- SearchImageSets
- DICOMImportJob starten

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Kennzeichnen eines HealthImaging Datenspeichers mithilfe eines SDK AWS

Die folgenden Codebeispiele zeigen, wie ein HealthImaging Datenspeicher mit Tags versehen wird.

Java

SDK für Java 2.x

So taggen Sie einen Datenspeicher.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
    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);
        }
    }
```

So listen Sie Tags für einen Datenspeicher auf.

```
                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());
                }
```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```
    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;
    }
```

So entfernen Sie Tags von einem Datenspeicher.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```
    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);
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

So taggen Sie einen Datenspeicher.

```
  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);
  }
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
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;
};
```

So listen Sie Tags für einen Datenspeicher auf.

```
  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);
  }
```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```
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;
};
```

So entfernen Sie Tags von einem Datenspeicher.

```
  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);
  }
```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```
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;
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

So taggen Sie einen Datenspeicher.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
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
```

So listen Sie Tags für einen Datenspeicher auf.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```
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"]
```

So entfernen Sie Tags von einem Datenspeicher.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```
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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Taggen eines HealthImaging Bilddatensatzes mithilfe eines SDK AWS

Die folgenden Codebeispiele zeigen, wie ein HealthImaging Bilddatensatz mit Tags versehen wird.

Java

SDK für Java 2.x

So taggen Sie einen Imagesatz.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
    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);
        }
    }
```

So listen Sie Tags für einen Imagesatz auf.

```
                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());
                }
```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```
    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;
    }
```

So entfernen Sie Tags von einem Imagesatz.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                Collections.singletonList("Deployment"));
```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```
    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);
        }
```

```
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

JavaScript

SDK für JavaScript (v3)

So taggen Sie einen Imagesatz.

```
  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);
  }
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
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;
};
```

So listen Sie Tags für einen Imagesatz auf.

```
  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);
  }
```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```
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;
};
```

So entfernen Sie Tags von einem Imagesatz.

```
  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);
  }
```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```
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;
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - ListTagsForResource
  - TagResource

- UntagResource

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Python

SDK für Python (Boto3)

So taggen Sie einen Imagesatz.

```
    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"})
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
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
```

So listen Sie Tags für einen Imagesatz auf.

```
    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)
```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```
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"]
```

So entfernen Sie Tags von einem Imagesatz.

```
    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"])
```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```
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
```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

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

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Service mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Überwachung von AWS HealthImaging

Überwachung und Protokollierung sind wichtige Bestandteile der Aufrechterhaltung der Sicherheit, Zuverlässigkeit, Verfügbarkeit und Leistung von AWS HealthImaging. AWS bietet die folgenden Protokollierungs- und Überwachungstools HealthImaging, mit denen Sie beobachten, melden können, wenn etwas nicht stimmt, und gegebenenfalls automatische Maßnahmen ergreifen können:

- AWS CloudTrailerfasst API-Aufrufe und zugehörige Ereignisse, die von oder im Namen Ihres AWS Kontos getätigt wurden, und übermittelt die Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket. Sie können feststellen, welche Benutzer und Konten angerufen wurden AWS, von welcher Quell-IP-Adresse aus die Anrufe getätigt wurden und wann die Aufrufe erfolgten. Weitere Informationen finden Sie im AWS CloudTrail -Benutzerhandbuch.
- Amazon CloudWatch überwacht Ihre AWS Ressourcen und die Anwendungen, auf denen Sie laufen, AWS in Echtzeit. Sie können Kennzahlen erfassen und verfolgen, benutzerdefinierte Dashboards erstellen und Alarme festlegen, die Sie benachrichtigen oder Maßnahmen ergreifen, wenn eine bestimmte Metrik einen von Ihnen festgelegten Schwellenwert erreicht. Sie können beispielsweise die CPU-Auslastung oder andere Kennzahlen Ihrer EC2 Amazon-Instances CloudWatch verfolgen und bei Bedarf automatisch neue Instances starten. Weitere Informationen finden Sie im CloudWatch Amazon-Benutzerhandbuch.
- Amazon EventBridge ist ein serverloser Event-Bus-Service, der es einfach macht, Ihre Anwendungen mit Daten aus einer Vielzahl von Quellen zu verbinden. EventBridge liefert einen Stream von Echtzeitdaten aus Ihren eigenen Anwendungen, Software-as-a-Service (SaaS-) Anwendungen und AWS Diensten und leitet diese Daten an Ziele wie Lambda weiter. Auf diese Weise können Sie Ereignisse überwachen, die in Services auftreten, und ereignisgesteuerte Architekturen erstellen. Weitere Informationen finden Sie im EventBridge Amazon-Benutzerhandbuch.

Themen

- Verwenden AWS CloudTrail mit HealthImaging
- Amazon verwenden CloudWatch mit HealthImaging
- Amazon verwenden EventBridge mit HealthImaging

Verwenden AWS CloudTrail mit HealthImaging

AWS HealthImaging ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Service in ausgeführt wurden HealthImaging. CloudTrail erfasst alle API-Aufrufe HealthImaging als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der HealthImaging Konsole und Codeaufrufen für die HealthImaging API-Operationen. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Übermittlung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für HealthImaging. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage ermitteln CloudTrail, an die die Anfrage gestellt wurde HealthImaging, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details.

Weitere Informationen CloudTrail dazu finden Sie im AWS CloudTrail Benutzerhandbuch.

Einen Trail erstellen

CloudTrail ist für Sie aktiviert AWS-Konto , wenn Sie das Konto erstellen. Wenn eine Aktivität in stattfindet HealthImaging, wird diese Aktivität zusammen mit anderen CloudTrail AWS Serviceereignissen im Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können in Ihrem AWS-Konto die neusten Ereignisse anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Ereignisse mit dem CloudTrail Ereignisverlauf anzeigen.

> **Note**
>
> Um den CloudTrail Ereignisverlauf für AWS HealthImaging in der anzuzeigen AWS-Managementkonsole, rufen Sie das Menü Suchattribute auf, wählen Sie Ereignisquelle aus und wählen Sie`medical-imaging.amazonaws.com`.

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem System AWS-Konto, einschließlich der Ereignisse für HealthImaging, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole anlegen, gilt dieser für alle AWS-Regionen-Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste

konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- In CloudTrail unterstützte Services und Integrationen
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

> **Note**
>
> AWS HealthImaging unterstützt zwei Arten von CloudTrail Ereignissen: Verwaltungsereignisse und Datenereignisse. Verwaltungsereignisse sind die allgemeinen Ereignisse, die jeder AWS Service generiert, darunter HealthImaging. Standardmäßig wird die Protokollierung bei jedem HealthImaging API-Aufruf, bei dem sie aktiviert ist, auf Verwaltungsereignisse angewendet. Datenereignisse sind fakturierbar und in der Regel für APIs Ereignisse mit hohen Transaktionsraten pro Sekunde (tps) reserviert. Aus Kostengründen können Sie die CloudTrail Protokollierung also deaktivieren.
> Mit HealthImaging Ausnahme von werden alle in der HealthImaging AWS-API-Referenz aufgeführten nativen API-Aktionen als Verwaltungsereignisse klassifiziert `GetImageFrame`. Die `GetImageFrame` Aktion ist CloudTrail als Datenereignis integriert und muss daher aktiviert werden. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen im Benutzerhandbuch für AWS CloudTrail .
> DICOMweb WADO-RS API-Aktionen werden in als Datenereignisse eingestuft. Sie müssen CloudTrail sich daher für sie anmelden. Weitere Informationen finden Sie unter DICOM-Daten werden abgerufen von HealthImaging und Protokollieren von Datenereignissen im AWS CloudTrail Benutzerhandbuch.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM-) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.

- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie im CloudTrail`userIdentity`Element.

Grundlegendes zu Protokolleinträgen

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass sie nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag für HealthImaging , der die `GetDICOMImportJob` Aktion demonstriert.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "XXXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
        "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/
ce6d90ba-5fba-4456-a7bc-f9bc877597c3"
        "accountId": "123456789012",
        "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "XXXXXXXXXXXXXXXXXXXXX",
                "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
                "accountId": "123456789012",
                "userName": "TestAccessRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-28T15:52:42Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-28T16:02:30Z",
    "eventSource": "medical-imaging.amazonaws.com",
```

```
    "eventName": "GetDICOMImportJob",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-
Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync
 http/Apache cfg/retry-mode/standard",
    "requestParameters": {
        "jobId": "5d08d05d6aab2a27922d6260926077d4",
        "datastoreId": "12345678901234567890123456789012"
    },
    "responseElements": null,
    "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
    "eventID": "26307f73-07f4-4276-b379-d362aa303b22",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "824333766656",
    "eventCategory": "Management"
}
```


Amazon verwenden CloudWatch mit HealthImaging

Sie können AWS HealthImaging mithilfe von AWS überwachen CloudWatch, das Rohdaten sammelt und zu lesbaren Metriken verarbeitet, die nahezu in Echtzeit verfügbar sind. Diese Statistiken werden 15 Monate lang aufbewahrt, sodass Sie diese historischen Informationen nutzen und sich einen besseren Überblick über die Leistung Ihrer Webanwendung oder Ihres Dienstes verschaffen können. Sie können auch Alarme einrichten, die auf bestimmte Grenzwerte achten und Benachrichtigungen senden oder Aktivitäten auslösen, wenn diese Grenzwerte erreicht werden. Weitere Informationen finden Sie im CloudWatch Amazon-Benutzerhandbuch.

HealthImaging veröffentlicht die folgenden Arten von CloudWatch Metriken im `AWS/HealthImaging` Namespace:

- API-Metriken — Anzahl der Aufrufe für HealthImaging API-Operationen
- HealthImaging Metriken — Datenspeicher und Ressourcennutzung auf Kontoebene

> **Note**
>
> - Für die meisten werden Metriken gemeldet. HealthImaging APIs

- HealthImaging Metriken sind nur für Datenspeicher verfügbar, die nach dem 9. Februar 2026 oder nach Einreichung eines Support-Falls erstellt wurden.

HealthImaging AWS-API-Metriken

In den folgenden Tabellen sind die Metriken und Dimensionen für aufgeführt HealthImaging. Jede Zahl wird als Häufigkeitszahl für einen vom Benutzer angegebenen Datenbereich dargestellt.

Metrik

Kennzahlen	Description
`CallCount`	Die Anzahl der Anrufe an APIs. Dies kann entweder für das Konto oder einen bestimmten Datenspeicher gemeldet werden. Einheiten: Anzahl Gültige Statistiken: Summe, Anzahl Dimensionen: Vorgang, Datenspeicher-ID, Datenspeichertyp

Sie können Metriken für HealthImaging mit der AWS-Managementkonsole AWS CLI, oder der CloudWatch API abrufen. Sie können die CloudWatch API über eines der Amazon AWS Software Development Kits (SDKs) oder die CloudWatch API-Tools verwenden. In der HealthImaging Konsole werden Diagramme angezeigt, die auf den Rohdaten der CloudWatch API basieren.

Sie müssen über die entsprechenden CloudWatch Berechtigungen für die Überwachung HealthImaging verfügen CloudWatch. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für CloudWatch im CloudWatch Benutzerhandbuch.

HealthImaging AWS-Metriken

Der `AWS/HealthImaging` Namespace umfasst die folgenden Metriken auf Konto- und Datenspeicherebene.

Metriken auf Kontoebene

Metriken auf Kontoebene bieten aggregierten Überblick über alle Datenspeicher in Ihrem Konto.

Metriken auf Kontoebene

Metrik	Description
`DataStoreCount`	Die Anzahl der Datenspeicher mit aktivem Status. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt
`ImageSetCount`	Die Gesamtzahl der Bilddatensätze in allen Datenspeichern. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt
`StorageBytes`	Die Datenmenge in Byte, die in allen Datenspeichern der folgenden Speicherebenen gespeichert ist: • Häufiger Zugriff (`FrequentAccessStorage`) • Sofortiger Zugriff archivieren (`ArchiveInstantAccessStorage`) Dieser Wert wird berechnet, indem die Größe aller Bilddatensätze in allen Datenspeichern summiert wird. Gültige Filter auf Speicherebene (siehe Dimension): `StorageTier` • `FrequentAccessStorage` • `ArchiveInstantAccessStorage`

Metrik	Description
	• `AllStorage` Einheiten: Byte Gültige Statistiken: Summe, Durchschnitt

Metriken auf Datenspeicherebene

Metriken auf Datenspeicherebene bieten einen detaillierten Einblick in einzelne Datenspeicher.

Metriken auf Datenspeicherebene

Metrik	Description
`TotalImageSetCount`	Die Gesamtzahl der Bildsätze im Datenspeicher. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt
`PrimaryImageSetCount`	Die Anzahl der primären Bilddatensätze im Datenspeicher. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt
`SmallImageSetCount`	Die Anzahl der Bildsätze im Datenspeicher beträgt weniger als 5 MB. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt
`StorageBytes`	Die Datenmenge in Byte, die in allen Datenspeichern in den folgenden Speicherebenen gespeichert ist:

Metrik	Description
	• Häufiger Zugriff (`FrequentAccessStorage`) • Sofortiger Zugriff archivieren (`ArchiveInstantAccessStorage`) Dieser Wert wird berechnet, indem die Größe aller Bilddatensätze im Datenspeicher summiert wird. Gültige Filter auf Speicherebene (siehe Dimension): `StorageTier` • FrequentAccessStorage • ArchiveInstantAccessStorage • AllStorage Einheiten: Byte Gültige Statistiken: Summe, Durchschnitt
`DICOMStudyCount`	Die Anzahl der DICOM-Studien im Datenspeicher. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt
`DICOMSeriesCount`	Die Anzahl der DICOM-Serien im Datenspeicher. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt

Metrik	Description
`DICOMInstanceCount`	Die Anzahl der DICOM-Instanzen im Datenspeicher. Einheiten: Anzahl Gültige Statistiken: Summe, Durchschnitt
`StructuredStorageBytes`	Die Menge an strukturiertem Speicher in Byte, die vom Datenspeicher indexiert wird. Einheiten: Byte Gültige Statistiken: Summe, Durchschnitt

HealthImaging Abmessungen in CloudWatch

Die folgenden Dimensionen werden zum Filtern von HealthImaging Metriken verwendet.

Dimensionen

Dimension	Description
`AccountId`	Diese Dimension filtert die Daten für das identifizierte AWS Konto.
`DatastoreId`	Diese Dimension filtert nur die Daten für den identifizierten Datenspeicher.
`StorageTier`	Diese Dimension filtert die Daten nach den folgenden Speicherebenen: • `FrequentAccessStorage` — Die Anzahl der Byte, die für Bilddatensätze im Frequenzzugriffsspeicher verwendet werden. • `ArchiveInstantAccessStorage` — Die Anzahl der Byte, die für Bilddaten

Dimension	Description
	sätze im Archive Instant Access-Speicher verwendet werden. • `AllStorage` — Die Gesamtzahl der Byte auf allen Speicherebenen.

Zugriff auf HealthImaging Metriken

Sie können wie folgt auf HealthImaging Metriken zugreifen:

- AWS-Managementkonsole- Metriken in der CloudWatch Konsole anzeigen
- AWS CLI- Verwenden Sie CloudWatch CLI-Befehle
- CloudWatch API — Zugriff über AWS SDKs unsere CloudWatch API-Tools

Sie müssen über die entsprechenden Berechtigungen für die Überwachung HealthImaging verfügen CloudWatch. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für CloudWatch im CloudWatch Benutzerhandbuch.

HealthImaging Metriken einrichten

Um HealthImaging Metriken in Ihrem CloudWatch Konto zu erhalten, müssen Sie eine dienstbezogene Rolle einrichten, die es Ihnen ermöglicht, Metriken in Ihrem Namen HealthImaging zu veröffentlichen. Weitere Informationen HealthImagingzum Erstellen einer dienstbezogenen Rolle finden Sie unter Verwenden von dienstbezogenen Rollen.

Metriken anzeigen HealthImaging

Um Metriken anzuzeigen (CloudWatch Konsole)

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die CloudWatchKonsole.
2. Wählen Sie Metriken, wählen Sie Alle Metriken und dann **`AWS/HealthImaging`**.
3. Wählen Sie die Dimension aus:
   - Nach Konto-ID — Metriken auf Kontoebene anzeigen
   - Nach Datenspeicher-ID — Metriken auf Datenspeicherebene anzeigen
4. Wählen Sie einen Metriknamen und dann Zur Grafik hinzufügen aus.

5. Wählen Sie einen Wert für den Datumsbereich, und die Anzahl der Metriken wird angezeigt.

Einen Alarm erstellen mit CloudWatch

Ein CloudWatch Alarm überwacht eine einzelne Metrik über einen bestimmten Zeitraum und führt eine oder mehrere Aktionen aus: das Senden einer Benachrichtigung an ein Amazon Simple Notification Service (Amazon SNS) -Thema oder an eine Auto Scaling Scaling-Richtlinie. Die Aktion oder Aktionen basieren auf dem Wert der Metrik im Verhältnis zu einem bestimmten Schwellenwert über eine von Ihnen angegebene Anzahl von Zeiträumen. CloudWatch kann Ihnen auch eine Amazon SNS SNS-Nachricht senden, wenn sich der Alarmstatus ändert.

CloudWatch Alarme lösen nur dann Aktionen aus, wenn sich der Status ändert und für den von Ihnen angegebenen Zeitraum andauert. Weitere Informationen finden Sie unter Alarme verwenden CloudWatch.

Amazon verwenden EventBridge mit HealthImaging

Amazon EventBridge ist ein serverloser Service, der Ereignisse verwendet, um Anwendungskomponenten miteinander zu verbinden, sodass Sie leichter skalierbare, ereignisgesteuerte Anwendungen erstellen können. Die Grundlage von EventBridge ist die Erstellung von Regeln, die Ereignisse an Ziele weiterleiten. AWS HealthImaging bietet eine dauerhafte Bereitstellung von Statusänderungen für EventBridge. Weitere Informationen finden Sie unter Was ist Amazon EventBridge? im EventBridge Amazon-Benutzerhandbuch.

Topics

- HealthImaging Ereignisse wurden gesendet an EventBridge
- HealthImaging Struktur und Beispiele für Ereignisse

HealthImaging Ereignisse wurden gesendet an EventBridge

In der folgenden Tabelle sind alle HealthImaging Ereignisse aufgeführt, die EventBridge zur Verarbeitung gesendet wurden.

HealthImaging Ereignistyp	Status
Ereignisse im Datenspeicher	

HealthImaging Ereignistyp	Status
Datenspeicher erstellen	`CREATING`
Die Erstellung des Datenspeichers ist fehlgeschlagen	`CREATE_FAILED`
Datenspeicher wurde erstellt	`ACTIVE`
Datenspeicher löschen	`DELETING`
Datenspeicher gelöscht	`DELETED`
Weitere Informationen finden Sie unter DataStoreStatus in der HealthImaging AWS-API-Referenz.	
Job-Ereignisse importieren	
Job importieren übermittelt	`SUBMITTED`
Job in Bearbeitung importieren	`IN_PROGRESS`
Importauftrag abgeschlossen	`COMPLETED`
Importjob ist fehlgeschlagen	`FAILED`
Weitere Informationen finden Sie unter JobStatus in der HealthImaging AWS-API-Referenz.	
Ereignisse im Bilddatensatz	
Bildsatz erstellt	`CREATED`
Bilddatensatz wird kopiert	`COPYING`
Kopieren von Bilddatensätzen mit schreibgeschütztem Zugriff	`COPYING_WITH_READ_ONLY_ACCESS`
Bilddatensatz wurde kopiert	`COPIED`
Das Kopieren des Bildsatzes ist fehlgeschlagen	`COPY_FAILED`

HealthImaging Ereignistyp	Status
Aktualisierung des Bildsatzes	`UPDATING`
Der Bildsatz wurde aktualisiert	`UPDATED`
Die Aktualisierung des Bildsatzes ist fehlgeschlagen	`UPDATE_FAILED`
Der Bildsatz wurde gelöscht	`DELETING`
Der Bildsatz wurde gelöscht	`DELETED`

Weitere Informationen finden Sie ImageSetWorkflowStatusin der HealthImaging AWS-API-Referenz.

HealthImaging Struktur und Beispiele für Ereignisse

HealthImaging Ereignisse sind Objekte mit JSON-Struktur, die auch Metadatendetails enthalten. Sie können die Metadaten als Eingabe verwenden, um entweder ein Ereignis neu zu erstellen oder weitere Informationen zu erhalten. Alle zugehörigen Metadatenfelder sind in einer Tabelle unter den Codebeispielen in den folgenden Menüs aufgeführt. Weitere Informationen finden Sie unter Referenz zur Event-Struktur im EventBridge Amazon-Benutzerhandbuch.

> **Note**
>
> Das `source` Attribut für HealthImaging Event-Strukturen ist`aws.medical-imaging`.

Ereignisse im Datenspeicher

Data Store Creating

Bundesstaat - **`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

Bundesstaat - **`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

Bundesstaat - **`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

Bundesstaat - **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

Bundesstaat - **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"
    }
}
```

Datenspeicher-Ereignisse — Metadaten-Beschreibungen

Name	Typ	Description
`version`	Zeichenfolge	Die Version des EventBridge Ereignisschemas.
`id`	Zeichenfolge	Die UUID der Version 4, die für jedes Ereignis generiert wurde.
`detail-type`	Zeichenfolge	Der Ereignistyp, der gesendet wird.
`source`	Zeichenfolge	Identifiziert den Service, aus dem das Ereignis stammt.
`account`	Zeichenfolge	Die 12-stellige AWS-Konto-ID des Datenspeicher-Besitzers.
`time`	Zeichenfolge	Der Zeitpunkt, zu dem das Ereignis eingetreten ist.
`region`	Zeichenfolge	Identifiziert die AWS Region des Datenspeichers.

Name	Typ	Description
`resources`	Array (Zeichenfolge)	Ein JSON-Array, das den ARN des Datenspeichers enthält.
`detail`	object	Ein JSON-Objekt, das Informationen zum Ereignis enthält.
`detail.imagingVersion`	Zeichenfolge	Die Versions-ID, die Änderungen HealthImaging am Ereignisdetailschema verfolgt.
`detail.datastoreId`	Zeichenfolge	Die Datenspeicher-ID, die dem Statusänderungsereignis zugeordnet ist.
`detail.datastoreName`	Zeichenfolge	Der Name des Datenspeichers.
`detail.datastoreStatus`	Zeichenfolge	Der aktuelle Status des Datenspeichers.

Job-Ereignisse importieren

Import Job Submitted

Bundesland - **`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

Bundesstaat - **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

Bundesstaat - **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

Bundesstaat - **`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/"
    }
}
```

Job-Ereignisse importieren — Metadaten-Beschreibungen

Name	Typ	Description
`version`	Zeichenfolge	Die Version des EventBridge Ereignisschemas.
`id`	Zeichenfolge	Die UUID der Version 4, die für jedes Ereignis generiert wurde.
`detail-type`	Zeichenfolge	Der Ereignistyp, der gesendet wird.
`source`	Zeichenfolge	Identifiziert den Service, aus dem das Ereignis stammt.
`account`	Zeichenfolge	Die 12-stellige AWS-Konto-ID des Datenspeicher-Besitzers.
`time`	Zeichenfolge	Der Zeitpunkt, zu dem das Ereignis eingetreten ist.
`region`	Zeichenfolge	Identifiziert die AWS Region des Datenspeichers.
`resources`	Array (Zeichenfolge)	Ein JSON-Array, das den ARN des Datenspeichers enthält.
`detail`	object	Ein JSON-Objekt, das Informationen zum Ereignis enthält.
`detail.imagingVersion`	Zeichenfolge	Die Versions-ID, die Änderungen HealthImaging am Ereignisdetailschema verfolgt.

Name	Typ	Description
`detail.datastoreId`	Zeichenfolge	Der Datenspeicher, der das Statusänderungsereignis generiert hat.
`detail.jobId`	Zeichenfolge	Die mit dem Statusänd erungsereignis verknüpfte Importjob-ID.
`detail.jobName`	Zeichenfolge	Der Name des Importauftrags.
`detail.jobStatus`	Zeichenfolge	Der aktuelle Jobstatus.
`detail.inputS3Uri`	Zeichenfolge	Der Eingabepräfixpfad für den S3-Bucket, der die zu importierenden DICOM-Dat eien enthält.
`detail.outputS3Uri`	Zeichenfolge	Das Ausgabepräfix des S3-Buckets, in den die Ergebniss e des DICOM-Importjobs hochgeladen werden.

Ereignisse im Bilddatensatz

Image Set Created

Bundesstaat - **`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

Bundesstaat - **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

Bundesstaat - **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

Bundesstaat - **`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

Bundesstaat - **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

Bundesstaat - **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

Bundesstaat - **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

Bundesstaat - **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

Bundesstaat - **`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

Bundesstaat - **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"
   }
}
```

Bilderdaten-Ereignisse — Metadaten-Beschreibungen

Name	Typ	Description
`version`	Zeichenfolge	Die Version des EventBridge Ereignisschemas.
`id`	Zeichenfolge	Die UUID der Version 4, die für jedes Ereignis generiert wurde.
`detail-type`	Zeichenfolge	Der Ereignistyp, der gesendet wird.
`source`	Zeichenfolge	Identifiziert den Service, aus dem das Ereignis stammt.

Name	Typ	Description
`account`	Zeichenfolge	Die 12-stellige AWS-Konto-ID des Datenspeicher-Besitzers.
`time`	Zeichenfolge	Der Zeitpunkt, zu dem das Ereignis eingetreten ist.
`region`	Zeichenfolge	Identifiziert die AWS Region des Datenspeichers.
`resources`	Array (Zeichenfolge)	Ein JSON-Array, das den ARN des Bildsatzes enthält.
`detail`	object	Ein JSON-Objekt, das Informationen zum Ereignis enthält.
`detail.imagingVersion`	Zeichenfolge	Die Versions-ID, die Änderungen HealthImaging am Ereignisdetailschema verfolgt.
`detail.isPrimary`	boolesch	Gibt an, ob die importierten Daten erfolgreich in der verwalteten Hierarchie organisiert wurden oder ob Metadatenkonflikte bestehen, die gelöst werden müssen.
`detail.imageSetVersion`	Zeichenfolge	Die Version des Bildsatzes wird inkrementiert, wenn eine Instanz mehr als einmal importiert wird. Die neueste Version überschreibt alle älteren Versionen, die in einem primären Image-Set gespeichert sind.

Name	Typ	Description
`detail.datastoreId`	Zeichenfolge	Die Datenspeicher-ID, die das Statusänderungsereignis generiert hat.
`detail.imagesetId`	Zeichenfolge	Die Bilddatensatz-ID, die dem Statusänderungsereignis zugeordnet ist.
`detail.imageSetState`	Zeichenfolge	Der aktuelle Status des Bilddatensatzes.
`detail.imageSetWorkflowStatus`	Zeichenfolge	Der aktuelle Workflow-Status des Bilddatensatzes.

Sicherheit in AWS HealthImaging

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von Rechenzentren und Netzwerkarchitekturen, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame AWS Verantwortung von Ihnen und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der AWS Dienste in der ausgeführt AWS Cloud werden. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für AWS gelten HealthImaging, finden Sie unter AWS Services im Bereich nach Compliance-Programm AWS .
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Service, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Nutzung anwenden können HealthImaging. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen HealthImaging , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer HealthImaging Ressourcen unterstützen.

Topics

- Datenschutz in AWS HealthImaging
- Identity and Access Management für AWS HealthImaging
- Konformitätsvalidierung für AWS HealthImaging
- Infrastruktursicherheit in AWS HealthImaging
- HealthImaging AWS-Ressourcen erstellen mit AWS CloudFormation
- AWS HealthImaging und Schnittstellen-VPC-Endpunkte ()AWS PrivateLink
- Kontoübergreifender Import für AWS HealthImaging
- Resilienz in AWS HealthImaging

Datenschutz in AWS HealthImaging

Das AWS Modell der gilt für den Datenschutz in AWS HealthImaging. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz . Informationen zum Datenschutz in Europa finden Sie im General Data Protection Regulation (GDPR) Center.

Aus Datenschutzgründen empfehlen wir, dass Sie Ihre AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit der Konsole, der API HealthImaging oder den SDKs arbeiten oder diese anderweitig AWS-Services verwenden. AWS CLI AWS Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet

werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Themen

- Datenverschlüsselung
- Datenschutz im Netzwerkverkehr

Datenverschlüsselung

Mit AWS HealthImaging können Sie Ihren in der Cloud gespeicherten Daten eine Sicherheitsebene hinzufügen und skalierbare und effiziente Verschlüsselungsfunktionen bereitstellen. Dazu zählen:

- Verschlüsselungsfunktionen für Daten im Ruhezustand sind in den meisten AWS Services verfügbar
- Flexible Schlüsselverwaltungsoptionen AWS Key Management Service, mit denen Sie wählen können, ob Sie die Verschlüsselungsschlüssel AWS verwalten möchten oder ob Sie die vollständige Kontrolle über Ihre eigenen Schlüssel behalten möchten.
- AWS eigene AWS KMS Verschlüsselungsschlüssel
- Verschlüsselte Nachrichtenwarteschlangen für die Übertragung sensibler Daten mit serverseitiger Verschlüsselung (SSE) für Amazon SQS

Darüber hinaus AWS bietet es APIs, mit denen Sie Verschlüsselung und Datenschutz in alle Services integrieren können, die Sie in einer Umgebung entwickeln oder bereitstellen. AWS

Verschlüsselung im Ruhezustand

HealthImaging Verschlüsselt gespeicherte Kundendaten standardmäßig mit einem Schlüssel, der dem Dienst gehört AWS Key Management Service . Optional können Sie so konfigurieren HealthImaging , dass Daten im Ruhezustand mithilfe eines symmetrischen, vom Kunden verwalteten AWS KMS Schlüssels verschlüsselt werden, den Sie selbst erstellen, besitzen und verwalten. Weitere Informationen finden Sie unter Erstellen eines KMS-Schlüssels mit symmetrischer Verschlüsselung im Entwicklerhandbuch.AWS Key Management Service

Verschlüsselung während der Übertragung

HealthImaging verwendet TLS 1.2, um Daten bei der Übertragung über den öffentlichen Endpunkt und über Backend-Services zu verschlüsseln.

Schlüsselverwaltung

AWS KMS Schlüssel (KMS-Schlüssel) sind die primäre Ressource in AWS Key Management Service. Sie können auch Datenschlüssel für die Verwendung außerhalb von generieren AWS KMS.

AWS eigener KMS-Schlüssel

HealthImaging verwendet diese Schlüssel standardmäßig, um potenziell vertrauliche Informationen wie personenbezogene Daten oder private Gesundheitsinformationen (PHI) im Ruhezustand automatisch zu verschlüsseln. AWS Eigene KMS-Schlüssel werden nicht in Ihrem Konto gespeichert. Sie sind Teil einer Sammlung von KMS-Schlüsseln, die AWS Eigentümer sind und für die Verwendung in mehreren AWS Konten verwaltet werden. AWS Dienste können AWS eigene KMS-Schlüssel verwenden, um Ihre Daten zu schützen. Sie können AWS eigene KMS-Schlüssel nicht anzeigen, verwalten, verwenden oder deren Verwendung überprüfen. Sie müssen jedoch keine Arbeit verrichten oder Programme ändern, um die Schlüssel zu schützen, mit denen Ihre Daten verschlüsselt werden.

Ihnen wird weder eine monatliche Gebühr noch eine Nutzungsgebühr berechnet, wenn Sie AWS eigene KMS-Schlüssel verwenden, und diese werden nicht auf die AWS KMS Kontingente für Ihr Konto angerechnet. Weitere Informationen finden Sie unter AWS-eigene Schlüssel im AWS Key Management Service Entwicklerhandbuch.

Kundenseitig verwaltete KMS-Schlüssel

Wenn Sie die volle Kontrolle über den AWS KMS Lebenszyklus und die Nutzung haben möchten, HealthImaging unterstützt es die Verwendung eines symmetrischen, vom Kunden verwalteten KMS-Schlüssels, den Sie selbst erstellen, besitzen und verwalten. Da Sie die volle Kontrolle über diese Verschlüsselungsebene haben, können Sie beispielsweise folgende Aufgaben ausführen:

- Einrichtung und Pflege wichtiger Richtlinien, IAM-Richtlinien und Zuschüsse
- Kryptographisches Material mit rotierendem Schlüssel
- Aktivieren und Deaktivieren wichtiger Richtlinien
- Hinzufügen von -Tags

- Erstellen von Schlüsselaliasen
- Schlüssel für das Löschen von Schlüsseln planen

Sie können es auch verwenden CloudTrail , um die Anfragen nachzuverfolgen, die in Ihrem Namen HealthImaging AWS KMS an gesendet werden. Es AWS KMS fallen zusätzliche Gebühren an. Weitere Informationen finden Sie im AWS Key Management Service Entwicklerhandbuch unter Vom Kunden verwaltete Schlüssel.

Einen vom Kunden verwalteten Schlüssel erstellen

Sie können einen symmetrischen, vom Kunden verwalteten Schlüssel mithilfe der AWS-Managementkonsole oder der AWS KMS APIs erstellen. Weitere Informationen finden Sie unter KMS-Schlüssel für symmetrische Verschlüsselung erstellen im AWS Key Management Service Entwicklerhandbuch.

Schlüsselrichtlinien steuern den Zugriff auf den vom Kunden verwalteten Schlüssel. Jeder vom Kunden verwaltete Schlüssel muss über genau eine Schlüsselrichtlinie verfügen, die aussagt, wer den Schlüssel wie verwenden kann. Wenn Sie Ihren kundenseitig verwalteten Schlüssel erstellen, können Sie eine Schlüsselrichtlinie angeben. Weitere Informationen finden Sie unter Verwalten des Zugriffs auf kundenverwaltete Schlüssel im Entwicklerhandbuch zum AWS Key Management Service .

Um Ihren vom Kunden verwalteten Schlüssel mit Ihren HealthImaging Ressourcen verwenden zu können, müssen kms: CreateGrant operations in der Schlüsselrichtlinie zulässig sein. Dadurch wird einem vom Kunden verwalteten Schlüssel ein Grant hinzugefügt, der den Zugriff auf einen bestimmten KMS-Schlüssel steuert, wodurch ein Benutzer Zugriff auf die für Grant-Operationen erforderlichen HealthImaging Zugriffsrechte erhält. Weitere Informationen finden Sie unter Grants AWS KMS im AWS Key Management Service Developer Guide.

Um Ihren vom Kunden verwalteten KMS-Schlüssel mit Ihren HealthImaging Ressourcen zu verwenden, müssen die folgenden API-Operationen in der Schlüsselrichtlinie zugelassen sein:

- `kms:DescribeKey`stellt die vom Kunden verwalteten Schlüsseldetails bereit, die zur Validierung des Schlüssels erforderlich sind. Dies ist für alle Operationen erforderlich.
- `kms:GenerateDataKey`bietet Zugriff auf verschlüsselte Ressourcen im Ruhezustand für alle Schreibvorgänge.
- `kms:Decrypt`bietet Zugriff auf Lese- oder Suchvorgänge für verschlüsselte Ressourcen.

- `kms:ReEncrypt*`bietet Zugriff auf neu verschlüsselte Ressourcen.

Im Folgenden finden Sie ein Beispiel für eine Richtlinienanweisung, die es einem Benutzer ermöglicht, einen Datenspeicher zu erstellen und mit diesem zu interagieren HealthImaging , in dem dieser Schlüssel verschlüsselt ist:

```
{
    "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"
        }
    }
}
```

Erforderliche IAM-Berechtigungen für die Verwendung eines vom Kunden verwalteten KMS-Schlüssels

Beim Erstellen eines Datenspeichers mit aktivierter AWS KMS Verschlüsselung mithilfe eines vom Kunden verwalteten KMS-Schlüssels sind für den Benutzer oder die Rolle, die den HealthImaging Datenspeicher erstellt, Berechtigungen sowohl für die Schlüsselrichtlinie als auch für die IAM-Richtlinie erforderlich.

Weitere Informationen zu wichtigen Richtlinien finden Sie unter Aktivieren von IAM-Richtlinien im AWS Key Management Service Entwicklerhandbuch.

Der IAM-Benutzer, die IAM-Rolle oder das AWS Konto, das Ihre Repositorys erstellt, muss über Berechtigungen für die unten stehende Richtlinie sowie über die erforderlichen Berechtigungen für AWS verfügen. HealthImaging

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:GenerateDataKey",
        "kms:RetireGrant",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f"
    }
  ]
}
```

Wie verwendet Grants HealthImaging in AWS KMS

HealthImaging erfordert einen Zuschuss für die Verwendung Ihres vom Kunden verwalteten KMS-Schlüssels. Wenn Sie einen Datenspeicher erstellen, der mit einem vom Kunden verwalteten KMS-Schlüssel verschlüsselt ist, HealthImaging erstellt in Ihrem Namen einen Zuschuss, indem es eine CreateGrantAnfrage an sendet AWS KMS. Grants in AWS KMS werden verwendet, um HealthImaging Zugriff auf einen KMS-Schlüssel in einem Kundenkonto zu gewähren.

Die Zuschüsse, die HealthImaging in Ihrem Namen gewährt werden, sollten nicht zurückgezogen oder zurückgezogen werden. Wenn Sie die Erteilung widerrufen oder zurückziehen, mit der Sie die HealthImaging Erlaubnis erhalten, die AWS KMS Schlüssel in Ihrem Konto zu verwenden, HealthImaging können Sie nicht auf diese Daten zugreifen, neue Bildgebungsressourcen verschlüsseln, die in den Datenspeicher übertragen werden, oder sie entschlüsseln, wenn sie abgerufen werden. Wenn Sie eine Genehmigung für widerrufen oder zurückziehen HealthImaging, wird die Änderung sofort wirksam. Um die Zugriffsrechte zu widerrufen, sollten Sie den Datenspeicher

löschen, anstatt die Gewährung zu widerrufen. Wenn ein Datenspeicher gelöscht wird, werden die Zuschüsse in Ihrem Namen HealthImaging zurückgezogen.

Überwachen Sie Ihre Verschlüsselungsschlüssel für HealthImaging

Sie können CloudTrail damit die Anfragen verfolgen, die in Ihrem Namen HealthImaging AWS KMS an gesendet werden, wenn Sie einen vom Kunden verwalteten KMS-Schlüssel verwenden. Die Protokolleinträge im CloudTrail Protokoll werden in dem `userAgent` Feld angezeigt`medical-imaging.amazonaws.com`, sodass die Anfragen von eindeutig unterschieden werden können HealthImaging.

Bei den folgenden Beispielen handelt es sich um CloudTrail Ereignisse für`CreateGrant`, `GenerateDataKeyDecrypt`, und `DescribeKey` zur Überwachung von AWS KMS Vorgängen, die aufgerufen werden, HealthImaging um auf Daten zuzugreifen, die mit Ihrem vom Kunden verwalteten Schlüssel verschlüsselt wurden.

Im Folgenden wird gezeigt, wie Sie `CreateGrant` den HealthImaging Zugriff auf einen vom Kunden bereitgestellten KMS-Schlüssel ermöglichen, sodass HealthImaging dieser KMS-Schlüssel zur Verschlüsselung aller gespeicherten Kundendaten verwendet werden kann.

Benutzer müssen keine eigenen Zuschüsse erstellen. HealthImaging erstellt in Ihrem Namen einen Zuschuss, indem Sie eine `CreateGrant` Anfrage an senden AWS KMS. Zuschüsse in AWS KMS werden verwendet, um HealthImaging Zugriff auf einen AWS KMS Schlüssel in einem Kundenkonto zu gewähren.

```
{
            "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
            "GrantId":
 "44e88bc45b769499ce5ec4abd5ecb27eeb3b178a4782452aae65fe885ee5ba20",
            "Name": "MedicalImagingGrantForQIDO_ebff634a-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"
                }
            }
        }
```

Die folgenden Beispiele zeigen, wie Sie sicherstellen können`GenerateDataKey`, dass der Benutzer vor dem Speichern über die erforderlichen Berechtigungen zum Verschlüsseln von Daten verfügt.

```
{
    "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"
}
```

Das folgende Beispiel zeigt, wie der `Decrypt` Vorgang HealthImaging aufgerufen wird, mit dem der gespeicherte verschlüsselte Datenschlüssel für den Zugriff auf die verschlüsselten Daten verwendet wird.

```
{
    "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"
}
```

Das folgende Beispiel zeigt, wie der `DescribeKey` Vorgang HealthImaging verwendet wird, um zu überprüfen, ob sich der AWS KMS Schlüssel im Besitz des AWS KMS Kunden in einem verwendbaren Zustand befindet, und um einem Benutzer bei der Fehlerbehebung zu helfen, falls er nicht funktioniert.

```
{
    "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"
```

```
}
```

Weitere Informationen

Die folgenden Ressourcen enthalten weitere Informationen zur Verschlüsselung von Daten im Ruhezustand und befinden sich im AWS Key Management Service Entwicklerhandbuch.

- AWS KMS -Konzepte
- Bewährte Sicherheitsmethoden für AWS KMS

Datenschutz im Netzwerkverkehr

Der Datenverkehr ist sowohl zwischen HealthImaging und lokalen Anwendungen als auch zwischen HealthImaging Amazon S3 geschützt. Der Verkehr zwischen HealthImaging und AWS Key Management Service verwendet standardmäßig HTTPS.

- AWS HealthImaging ist ein regionaler Service, der in den Regionen USA Ost (Nord-Virginia), USA West (Oregon), Europa (Irland) und Asien-Pazifik (Sydney) verfügbar ist.
- Für den Datenverkehr zwischen HealthImaging und Amazon S3-Buckets verschlüsselt Transport Layer Security (TLS) Objekte bei der Übertragung zwischen Amazon S3 HealthImaging und zwischen Kundenanwendungen, die darauf zugreifen, sollten Sie nur verschlüsselte Verbindungen über HTTPS (TLS) zulassen, indem Sie die IAM-Richtlinien für `aws:SecureTransport condition`Amazon S3 S3-Buckets verwenden. HealthImaging Obwohl HealthImaging derzeit der öffentliche Endpunkt für den Zugriff auf Daten in Amazon S3 S3-Buckets verwendet wird, bedeutet dies nicht, dass die Daten das öffentliche Internet durchqueren. Der gesamte Datenverkehr zwischen Amazon S3 HealthImaging und Amazon S3 wird über das AWS Netzwerk geleitet und mit TLS verschlüsselt.

Identity and Access Management für AWS HealthImaging

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. HealthImaging IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe
- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- So HealthImaging arbeitet AWS mit IAM
- Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging
- AWS verwaltete Richtlinien für AWS HealthImaging
- Dienstübergreifende verwirrte Stellvertreterprävention in HealthImaging
- Verwenden von serviceverknüpften Rollen für HealthImaging
- Fehlerbehebung bei HealthImaging AWS-Identität und Zugriff

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe Fehlerbehebung bei HealthImaging AWS-Identität und Zugriff).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe So HealthImaging arbeitet AWS mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen aus einer Identitätsquelle zugreift. Verbundene Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu können.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind)

sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

So HealthImaging arbeitet AWS mit IAM

Bevor Sie IAM zur Verwaltung des Zugriffs auf verwenden, sollten Sie sich darüber informieren HealthImaging, mit welchen IAM-Funktionen Sie arbeiten können. HealthImaging

IAM-Funktionen, die Sie mit AWS verwenden können HealthImaging

IAM-Feature	HealthImaging Unterstützung
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Nein
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Richtlinienbedingungsschlüssel (services pezifisch)	Ja
ACLs	Nein
ABAC (Tags in Richtlinien)	Teilweise
Temporäre Anmeldeinformationen	Ja
Prinzipalberechtigungen	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Ja

Einen allgemeinen Überblick darüber, wie HealthImaging und andere AWS Dienste mit den meisten IAM-Funktionen funktionieren, finden Sie im IAM-Benutzerhandbuch unter AWS Dienste, die mit IAM funktionieren.

Identitätsbasierte Richtlinien für HealthImaging

Unterstützt Richtlinien auf Identitätsbasis: Ja

HealthImaging unterstützt Zustandsanweisungen, in denen die DICOM-Studie und die DICOM-Serie spezifiziert sind UIDs, sodass die Zugriffskontrolle auf eine oder mehrere DICOM-Studien oder -Serien beschränkt ist.

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien für HealthImaging

Beispiele für HealthImaging identitätsbasierte Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging

Ressourcenbasierte Richtlinien finden Sie in HealthImaging

Unterstützt ressourcenbasierte Richtlinien: Nein

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Richtlinienaktionen für HealthImaging

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Eine Liste der HealthImaging Aktionen finden Sie unter Von AWS definierte Aktionen HealthImaging in der Service Authorization Reference.

Bei Richtlinienaktionen wird vor der Aktion das folgende Präfix HealthImaging verwendet:

```
medical-imaging
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

```
"Action": [
      "medical-imaging:action1",
      "medical-imaging:action2"
         ]
```

Beispiele für HealthImaging identitätsbasierte Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging

Politische Ressourcen für HealthImaging

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen

(ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Eine Liste der HealthImaging Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter Von AWS definierte Ressourcentypen HealthImaging in der Service Authorization Reference. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen ARN verwenden können, finden Sie unter Von AWS definierte Aktionen HealthImaging.

Beispiele für HealthImaging identitätsbasierte Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging

Bedingungsschlüssel für Richtlinien für HealthImaging

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Eine Liste der HealthImaging Bedingungsschlüssel finden Sie unter Bedingungsschlüssel für AWS HealthImaging in der Service Authorization Reference. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Von AWS definierte Aktionen HealthImaging.

Beispiele für HealthImaging identitätsbasierte Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging

ACLs in HealthImaging

Unterstützt ACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

RBAC mit HealthImaging

Unterstützt RBAC	Ja

Das herkömmliche in IAM verwendete Autorisierungsmodell wird als rollenbasierte Zugriffskontrolle (RBAC, Role-Based Access Control) bezeichnet. RBAC definiert Berechtigungen basierend auf der Job-Funktion einer Person, die außerhalb von AWS als Rollebezeichnet wird. Weitere Informationen finden Sie unter Vergleich von ABAC mit dem traditionellen RBAC-Modell im IAM-Benutzerhandbuch.

ABAC mit HealthImaging

Unterstützt ABAC (Tags in Richtlinien): Teilweise

Warning

ABAC wird nicht über die API-Aktion erzwungen. `SearchImageSets` Jeder, der Zugriff auf die `SearchImageSets` Aktion hat, kann auf alle Metadaten für Bilddatensätze in einem Datenspeicher zugreifen.

Note

Bildsätze sind eine untergeordnete Ressource von Datenspeichern. Um ABAC verwenden zu können, muss ein Bildsatz dasselbe Tag wie ein Datenspeicher haben. Weitere Informationen finden Sie unter Ressourcen mit AWS taggen HealthImaging.

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen, auch als Tags bezeichnet, definiert werden. Sie können Tags an IAM-Entitäten und AWS -Ressourcen anhängen und dann ABAC-Richtlinien entwerfen, die Operationen zulassen, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter Definieren von Berechtigungen mit ABAC-Autorisierung im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe Attributbasierte Zugriffskontrolle (ABAC) verwenden im IAM-Benutzerhandbuch.

Verwenden temporärer Anmeldeinformationen mit HealthImaging

Unterstützt temporäre Anmeldeinformationen: Ja

Temporäre Anmeldeinformationen ermöglichen kurzfristigen Zugriff auf AWS Ressourcen und werden automatisch erstellt, wenn Sie einen Verbund verwenden oder die Rollen wechseln. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporäre Anmeldeinformationen in IAM und AWS-Services , die mit IAM funktionieren im IAM-Benutzerhandbuch.

Serviceübergreifende Prinzipalberechtigungen für HealthImaging

Unterstützt Forward Access Sessions (FAS): Ja

Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Richtlinien erteilen einem Prinzipal-Berechtigungen. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service auslösen.
In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen.
Informationen darüber, ob eine Aktion zusätzliche abhängige Aktionen in einer Richtlinie erfordert, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für AWS HealthImaging in der Service Authorization Reference.

Servicerollen für HealthImaging

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine IAM-Rolle, die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern

und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

> **Warning**
>
> Durch das Ändern der Berechtigungen für eine Servicerolle kann die HealthImaging Funktionalität beeinträchtigt werden. Bearbeiten Sie Servicerollen nur, HealthImaging wenn Sie dazu eine Anleitung erhalten.

Dienstbezogene Rollen für HealthImaging

Unterstützt serviceverknüpfte Rollen: Ja

Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

HealthImaging verwendet dienstbezogene Rollen, um CloudWatch Kennzahlen in Ihrem Konto zu veröffentlichen. Details zum Erstellen oder Verwalten von serviceverknüpften Rollen finden Sie unter AWS -Services, die mit IAM funktionieren.

Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging

Benutzer und Rollen haben standardmäßig nicht die Berechtigung, HealthImaging-Ressourcen zu erstellen oder zu ändern. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von IAM-Richtlinien (Konsole) im IAM-Benutzerhandbuch.

Einzelheiten zu den von Awesome definierten Aktionen und Ressourcentypen, einschließlich des Formats ARNs für die einzelnen Ressourcentypen, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für AWS Awesome in der Service Authorization Reference.

Themen

- Best Practices für Richtlinien
- Verwenden der Konsole HealthImaging

- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer
- Erteilung von Berechtigungen auf der Grundlage der Study Instance UID und der Series Instance UID

Best Practices für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand HealthImaging Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien oder Von AWS verwaltete Richtlinien für Auftragsfunktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON)

und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.

- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Best Practices für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwenden der Konsole HealthImaging

Um auf die HealthImaging AWS-Konsole zugreifen zu können, benötigen Sie ein Mindestmaß an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den HealthImaging Ressourcen in Ihrem aufzulisten und einzusehen AWS-Konto. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die die Benutzer ausführen möchten.

Um sicherzustellen, dass Benutzer und Rollen die HealthImaging Konsole weiterhin verwenden können, fügen Sie den Entitäten auch die HealthImaging *ConsoleAccess* oder die *ReadOnly* AWS verwaltete Richtlinie hinzu. Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der AWS CLI AWS OR-API.

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

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Erteilung von Berechtigungen auf der Grundlage der Study Instance UID und der Series Instance UID

HealthImaging DICOMWeb APIs unterstützt die Gewährung des Zugriffs auf Bilddatensätze auf der Grundlage der Study Instance UID und der Series Instance UID. Sie können IAM-Richtlinien definieren, die den Zugriff einschränken, indem Sie Bedingungsanweisungen mit den Kontextschlüsseln `StudyInstanceUID` und `SeriesInstanceUID` condition hinzufügen.

HealthImaging DICOMWeb APIs die `StudyInstanceUID` als erforderlichen Parameter verwenden, unterstützen IAM-Richtlinien, die den Zugriff auf der `StudyInstanceUID` Grundlage des Schlüssels einschränken. In ähnlicher Weise unterstützen diejenigen HealthImaging DICOMWeb

APIs , die `SeriesInstanceUID` als erforderlichen Parameter verwenden, Richtlinien mit dem `SeriesInstanceUID` Schlüssel.

HealthImaging APIs die IAM-Richtlinien mithilfe von **`StudyInstanceUID`** **`SeriesInstanceUID`** Kontextschlüsseln unterstützen

Name	Support für den **`StudyInstanceUID`** Zustand	Support für den **`SeriesInstanceUID`** Zustand
`GetDICOMInstance`	Ja	Ja
`GetDICOMInstanceFrames`	Ja	Ja
`GetDICOMInstanceMetadata`	Ja	Ja
`GetDICOMSeriesMetadata`	Ja	Ja
`GetDICOMBulkdata`	Ja	Ja
`SearchDICOMSeries`	Ja	Nein
`SearchDICOMInstances`	Ja	Ja
`StoreDICOMStudy`	Ja	Nein

> **Note**
>
> Eine HealthImaging API, die diesen Kontextschlüssel nicht unterstützt, funktioniert so, als ob kein Kontextschlüssel angegeben worden wäre, wenn sie mit einer Richtlinie aufgerufen wird, die einen `StudyInstanceUID` oder `SeriesInstanceUID` -Kontextschlüssel enthält.

Beispiel 1: Zugriff auf der Grundlage einer StudyInstance UID gewähren

Um nur bestimmten DICOM-Studien Zugriff zu gewähren, fügen Sie der Rolle eine Richtlinie hinzu, die eine Bedingung für die festlegt. `StudyInstanceUID`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMSeries"
            ],
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:StudyInstanceUID": "your study instance UID"
                }
            }
        }
    ]
}
```

Wenn diese Rolle übernommen wird`sts assume-role`, ist der Anrufer nur berechtigt, auf Bilddatensätze zuzugreifen, die der in der Rollenrichtlinie angegebenen Bedingung entsprechen. Andernfalls werden die Aufrufe abgewiesen und es wird ein `AccessDenied` Fehler ausgegeben. In diesem Fall erhält der Aufrufer Zugriff auf alle Bilddatensätze mit den angegebenen Werten. `StudyInstanceUID`

Sie können in Ihren Richtlinien alle IAM-String-Bedingungsoperatoren verwenden, einschließlich Platzhalterabgleich und Mehrfachübereinstimmungen.

Eine Beispielrichtlinie für den Platzhalterabgleich:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMSeries"
            ],
```

```
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-
id"
            ],
            "Condition": {
                "StringLike": {
                    "medical-imaging:StudyInstanceUID": "123.456.789*"
                }
            }
        }
    ]
}
```

Eine Beispielrichtlinie für mehrere Treffer:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMSeries"
            ],
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-
id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:StudyInstanceUID": [
                        "123.456.789",
                        "1.2.3.4.5.6"
                    ]
                }
            }
        }
    ]
}
```

Beispiel 2: Zugriff auf der Grundlage einer SeriesInstance UID gewähren

Um Zugriff nur auf bestimmte Bilddatensätze zu gewähren, die einer DICOM-Serie entsprechen, fügen Sie der Rolle eine Richtlinie hinzu, die eine Bedingung für die festlegt. `SeriesInstanceUID`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMInstances"
            ],
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:SeriesInstanceUID": [
                        "123.456.789",
                        "1.2.3.4.5.6"
                    ]
                }
            }
        }
    ]
}
```

Beispiel 3: Zugriff auf der Grundlage von und gewähren StudyInstance UIDs SeriesInstance UIDs

Um nur Zugriff auf Bilddatensätze einer bestimmten DICOM-Studie und -Serie zu gewähren, fügen Sie der Rolle eine Richtlinie hinzu, die Bedingungen sowohl für als auch festlegt. `StudyInstanceUID` `SeriesInstanceUID`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
```

```
                "medical-imaging:SearchDICOMInstances"
            ],
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-
id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:StudyInstanceUID": ["123.456.789"],
                    "medical-imaging:SeriesInstanceUID": ["1.2.3.4.5.6"]
                }
            }
        }
    ]
}
```

AWS verwaltete Richtlinien für AWS HealthImaging

Eine AWS verwaltete Richtlinie ist eine eigenständige Richtlinie, die von erstellt und verwaltet wird AWS. AWS Verwaltete Richtlinien dienen dazu, Berechtigungen für viele gängige Anwendungsfälle bereitzustellen, sodass Sie damit beginnen können, Benutzern, Gruppen und Rollen Berechtigungen zuzuweisen.

Beachten Sie, dass AWS verwaltete Richtlinien für Ihre speziellen Anwendungsfälle möglicherweise keine Berechtigungen mit den geringsten Rechten gewähren, da sie für alle AWS Kunden verfügbar sind. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind.

Sie können die in AWS verwalteten Richtlinien definierten Berechtigungen nicht ändern. Wenn die in einer AWS verwalteten Richtlinie definierten Berechtigungen AWS aktualisiert werden, wirkt sich das Update auf alle Prinzidentitäten (Benutzer, Gruppen und Rollen) aus, denen die Richtlinie zugeordnet ist. AWS aktualisiert eine AWS verwaltete Richtlinie höchstwahrscheinlich, wenn eine neue Richtlinie eingeführt AWS-Service wird oder neue API-Operationen für bestehende Dienste verfügbar werden.

Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien im IAM-Benutzerhandbuch.

Themen

- AWS verwaltete Richtlinie: AWSHealth ImagingServiceRolePolicy
- AWS verwaltete Richtlinie: AWSHealth ImagingFullAccess
- AWS verwaltete Richtlinie: AWSHealth ImagingReadOnlyAccess
- HealthImaging Aktualisierungen der verwalteten AWS Richtlinien

AWS verwaltete Richtlinie: AWSHealth ImagingServiceRolePolicy

Diese Richtlinie ist einer dienstbezogenen Rolle `AWSServiceRoleForHealthImaging` zugeordnet. Sie gewährt Berechtigungen für die Verwaltung von HealthImaging Servicebetrieben und die Veröffentlichung von Servicemetriken.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie AWSHealthImagingServiceRolePolicyim AWS Managed Policy Reference Guide.

AWS verwaltete Richtlinie: AWSHealth ImagingFullAccess

Sie können die `AWSHealthImagingFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt Administratorberechtigungen für alle HealthImaging Aktionen.

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 verwaltete Richtlinie: AWSHealth ImagingReadOnlyAccess

Sie können die `AWSHealthImagingReadOnlyAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt bestimmten HealthImaging AWS-Aktionen nur Leseberechtigungen.

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": "*"
    }]
}
```

HealthImaging Aktualisierungen der verwalteten AWS Richtlinien

Hier finden Sie Informationen zu Aktualisierungen AWS verwalteter Richtlinien HealthImaging seit Beginn der Nachverfolgung dieser Änderungen durch diesen Dienst. Abonnieren Sie den RSS-Feed auf der Seite Veröffentlichungen, um automatische Benachrichtigungen über Änderungen an dieser Seite zu erhalten.

Änderungen	Beschreibung	Date
AWSHealthImagingServiceRolePolicy	AWS HealthImaging hat eine neue verwaltete Richtlinie für die serviceverknüpfte Rolle hinzugefügt, die Berechtigungen für HealthImaging die Verwaltung von Servicebetrieben und die Veröffentlichung von Servicemetriken bietet.	9. Februar 2026
HealthImaging hat begonnen, Änderungen zu verfolgen	HealthImaging hat begonnen, Änderungen für die AWS verwalteten Richtlinien zu verfolgen.	19. Juli 2023

Dienstübergreifende verwirrte Stellvertreterprävention in HealthImaging

Das Problem des verwirrten Stellvertreters ist ein Sicherheitsproblem, bei dem eine Entität, die keine Berechtigung zur Durchführung einer Aktion hat, eine privilegiertere Entität zur Durchführung der Aktion zwingen kann. In AWS kann ein serviceübergreifender Identitätswechsel zum Problem des verwirrten Stellvertreters führen. Ein dienstübergreifender Identitätswechsel kann auftreten, wenn ein Dienst (der Anruf-Dienst) einen anderen Dienst anruft (den aufgerufenen Dienst). Der Anruf-Dienst kann so manipuliert werden, dass er seine Berechtigungen verwendet, um auf die Ressourcen eines

anderen Kunden zu reagieren, auf die er sonst nicht zugreifen dürfte. Um dies zu verhindern, bietet AWS Tools, mit denen Sie Ihre Daten für alle Services mit Service Principals schützen können, denen Zugriff auf Ressourcen in Ihrem Konto gewährt wurde.

Wir empfehlen, die Kontextschlüssel `aws:SourceArn`und die `aws:SourceAccount`globalen Bedingungsschlüssel in Ihren `ImportJobDataAccessRole` IAM-Richtlinien für Vertrauensbeziehungen zu verwenden, um die Berechtigungen einzuschränken, die AWS einem anderen Service für Ihre Ressource HealthImaging erteilt. Verwenden Sie `aws:SourceArn`, um nur einer Ressource den serviceübergreifenden Zugriff zuzuordnen. Verwenden Sie `aws:SourceAccount`, um jede Ressource in diesem Konto der serviceübergreifenden Nutzung zuzuordnen. Wenn Sie beide Kontextschlüssel für globale Bedingungen verwenden, müssen der `aws:SourceAccount` Wert und das Konto, auf das im `aws:SourceArn` Wert verwiesen wird, dieselbe Konto-ID verwenden, wenn sie in derselben Richtlinienerklärung verwendet werden.

Der Wert von `aws:SourceArn` muss der ARN des betroffenen Datenspeichers sein. Wenn Sie den vollständigen ARN des Datenspeichers nicht kennen oder wenn Sie mehrere Datenspeicher angeben, verwenden Sie den `aws:SourceArn` globalen Kontextbedingungsschlüssel mit dem Platzhalter * für die unbekannten Teile des ARN. Sie können beispielsweise festlegen `aws:SourceArn` auf`arn:aws:medical-imaging:us-west-2:111122223333:datastore/*`.

Im folgenden Beispiel für eine Vertrauensrichtlinie verwenden wir den `aws:SourceAccount` Bedingungsschlüssel `aws:SourceArn` und, um den Zugriff auf den Dienstprinzipal auf der Grundlage des ARN des Datenspeichers einzuschränken, um das Problem des verwirrten Stellvertreters zu vermeiden.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "medical-imaging.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/*"
```

```
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Verwenden von serviceverknüpften Rollen für HealthImaging

AWS HealthImaging verwendet AWS Identity and Access Management (IAM) -Serviceverknüpfte Rollen, die vom Service vordefiniert sind und alle Berechtigungen enthalten, die der Service benötigt, um andere AWS-Services in Ihrem Namen aufzurufen. Weitere Informationen finden Sie unter Berechtigungen für serviceverknüpfte Rollen im IAM-Benutzerhandbuch.

Berechtigungen für serviceverknüpfte Rollen HealthImaging

HealthImaging verwendet die angegebene dienstverknüpfte Rolle`AWSServiceRoleForHealthImaging`, um Vorgänge in Ihrem AWS Konto auszuführen. Sie müssen diese dienstbezogene Rolle erstellen, wenn Sie Messwerte HealthImaging zu Ihren Datenspeichern veröffentlichen möchten. CloudWatch

Die genannte Richtlinie für Rollenberechtigungen `AWSHealthImagingServiceRolePolicy` gewährt Berechtigungen für die Verwaltung von HealthImaging Servicebetrieben und die Veröffentlichung von Servicemetriken.

Informationen zu verwalteten Richtlinienaktualisierungen finden Sie unter HealthImaging Verwaltete Richtlinien.

Erstellen einer dienstbezogenen Rolle für HealthImaging

Erstellen Sie mit der IAM-Konsole eine serviceverknüpfte Rolle

Sie können mithilfe der IAM-Konsole eine serviceverknüpfte Rolle erstellen, indem Sie den Entitätstyp Vertrauenswürdig auswählen `AWS Service` und dann `HealthImaging` im Dropdownmenü „Anwendungsfall“ auswählen.

Erstellen Sie eine serviceverknüpfte Rolle mit der AWS-CLI

Führen Sie in der AWS-CLI Folgendes aus `aws iam create-service-linked-role –aws-service-name medical-imaging.amazonaws.com`

Löschen einer serviceverknüpften Rolle für HealthImaging

Sie können eine dienstverknüpfte Rolle jederzeit löschen. Dadurch werden jedoch keine Aktionen mehr in Ihrem AWS Konto ausgeführt, z. B. das Veröffentlichen HealthImaging von Data-Store-Metriken in. CloudWatch

So löschen Sie die serviceverknüpfte Rolle mit IAM

Sie können die IAM-Konsole, die AWS-CLI oder die AWS-API verwenden, um die `AWSServiceRoleForHealthImaging` serviceverknüpfte Rolle zu löschen. Weitere Informationen finden Sie unter Löschen einer serviceverknüpften Rolle im IAM-Benutzerhandbuch. Wenn Sie eine serviceverknüpfte Rolle gelöscht haben, können Sie den Prozess zur Rollenerstellung verwenden, um eine neue zu erstellen.

Fehlerbehebung bei HealthImaging AWS-Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit HealthImaging und IAM auftreten können.

Themen

- Ich bin nicht berechtigt, eine Aktion durchzuführen in HealthImaging
- Ich bin nicht berechtigt, iam auszuführen: PassRole
- Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine HealthImaging Ressourcen ermöglichen

Ich bin nicht berechtigt, eine Aktion durchzuführen in HealthImaging

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, über die Konsole Details zu einer fiktiven *`my-example-widget`*-Ressource anzuzeigen, jedoch nicht über `AWS:`*`GetWidget`*-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 AWS:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `AWS:GetWidget`-Aktion auf die `my-example-widget`-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an HealthImaging übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in HealthImaging auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine HealthImaging Ressourcen ermöglichen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob diese Funktionen HealthImaging unterstützt werden, finden Sie unter. So HealthImaging arbeitet AWS mit IAM
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Konformitätsvalidierung für AWS HealthImaging

Externe Prüfer bewerten die Sicherheit und Konformität von AWS im HealthImaging Rahmen mehrerer AWS Compliance-Programme. Dazu gehört auch HIPAA. HealthImaging

Eine Liste der AWS Services im Rahmen bestimmter Compliance-Programme finden Sie unter AWS-Services in Umfang nach Compliance-Programm . Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen in AWS Artifact.

Ihre Compliance-Verantwortung bei der Nutzung von AWS HealthImaging hängt von der Sensibilität Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS bietet die folgenden Ressourcen zur Unterstützung bei der Einhaltung von Vorschriften:

- AWS Partnerlösungen — In den automatisierten Bereitstellungsleitfäden für Sicherheit und Compliance werden architektonische Überlegungen erörtert und Schritte zur Implementierung von sicherheits- und Compliance-orientierten Basisumgebungen beschrieben. AWS
- Whitepaper „Architecting for HIPAA Security and Compliance“ — In diesem Whitepaper wird beschrieben, wie Unternehmen HIPAA-konforme Anwendungen entwickeln können. AWS

- GxP Systems on AWS — Dieses Whitepaper enthält Informationen darüber, wie GxP-bezogene Compliance und Sicherheit angegangen wird, und bietet Anleitungen AWS zur Nutzung von Services im Kontext von GxP. AWS
- AWS Ressourcen zur Einhaltung von Vorschriften — Diese Sammlung von Arbeitsmappen und Leitfäden kann auf Ihre Branche und Ihren Standort zutreffen.
- Ressourcen anhand von Regeln bewerten — AWS Config bewertet, wie gut Ihre Ressourcenkonfigurationen internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub CSPM— Dieser AWS Service bietet einen umfassenden Überblick über Ihren Sicherheitsstatus und hilft Ihnen AWS , die Einhaltung der Sicherheitsstandards und bewährten Verfahren der Branche zu überprüfen.

Infrastruktursicherheit in AWS HealthImaging

Als verwalteter Service HealthImaging ist AWS durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die im Whitepaper Amazon Web Services: Sicherheitsprozesse im Überblick beschrieben sind.

Sie verwenden AWS veröffentlichte API-Aufrufe, um HealthImaging über das Netzwerk darauf zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.3 oder höher unterstützen. Clients müssen außerdem Verschlüsselungssammlungen mit PFS (Perfect Forward Secrecy) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) unterstützen. Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Sie können die AWS -Security-Token-Service(AWS STS) auch verwenden, um temporäre Sicherheitsanmeldeinformationen zum Signieren von Anfragen zu generieren.

HealthImaging AWS-Ressourcen erstellen mit AWS CloudFormation

AWS HealthImaging ist in einen Service integriert AWS CloudFormation, der Sie bei der Modellierung und Einrichtung Ihrer AWS Ressourcen unterstützt, sodass Sie weniger Zeit mit der Erstellung und Verwaltung Ihrer Ressourcen und Infrastruktur verbringen müssen. Sie erstellen eine Vorlage, die alle gewünschten AWS Ressourcen beschreibt und diese Ressourcen für Sie CloudFormation bereitstellt und konfiguriert.

Wenn Sie sie verwenden CloudFormation, können Sie Ihre Vorlage wiederverwenden, um Ihre HealthImaging Ressourcen konsistent und wiederholt einzurichten. Beschreiben Sie Ihre Ressourcen einmal und stellen Sie dann dieselben Ressourcen immer wieder in mehreren AWS-Konten Regionen bereit.

HealthImaging und CloudFormation Vorlagen

Um Ressourcen für und zugehörige Dienste bereitzustellen HealthImaging und zu konfigurieren, müssen Sie sich mit CloudFormation Vorlagen auskennen. Vorlagen sind formatierte Textdateien in JSON oder YAML. Diese Vorlagen beschreiben die Ressourcen, die Sie in Ihren CloudFormation Stacks bereitstellen möchten. Wenn Sie mit JSON oder YAML nicht vertraut sind, können Sie CloudFormation Designer verwenden, um Ihnen die ersten Schritte mit Vorlagen zu erleichtern. CloudFormation Weitere Informationen finden Sie unter Was ist CloudFormation -Designer? im AWS CloudFormation -Benutzerhandbuch.

AWS HealthImaging unterstützt die Erstellung von Datenspeichern mit CloudFormation. Weitere Informationen, einschließlich Beispielen für JSON- und YAML-Vorlagen für die Bereitstellung von HealthImaging Datenspeichern, finden Sie in der HealthImaging AWS-Ressourcentyp-Referenz im AWS CloudFormation Benutzerhandbuch.

Erfahren Sie mehr über CloudFormation

Weitere Informationen CloudFormation dazu finden Sie in den folgenden Ressourcen:

- AWS CloudFormation
- AWS CloudFormation Benutzerhandbuch
- CloudFormation API Referenz
- AWS CloudFormation Benutzerhandbuch für die Befehlszeilenschnittstelle

AWS HealthImaging und Schnittstellen-VPC-Endpunkte ()AWS PrivateLink

Sie können eine private Verbindung zwischen Ihrer VPC herstellen und AWS HealthImaging einen VPC-Schnittstellen-Endpunkt erstellen. Schnittstellenendpunkte werden mit einer Technologie betrieben AWS PrivateLink, mit der Sie privat HealthImaging APIs ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder AWS Direct Connect Connect-Verbindung zugreifen können. Instances

in Ihrer VPC benötigen keine öffentlichen IP-Adressen für die Kommunikation. HealthImaging APIs Datenverkehr zwischen Ihrer VPC und HealthImaging verlässt das Amazon-Netzwerk nicht.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere Elastic-Network-Schnittstellen in Ihren Subnetzen dargestellt.

Weitere Informationen finden Sie unter Interface VPC Endpoints (AWS PrivateLink) im Amazon VPC-Benutzerhandbuch.

Topics

- Überlegungen zu HealthImaging VPC-Endpunkten
- Erstellen eines Schnittstellen-VPC-Endpunkts für HealthImaging
- Erstellen einer VPC-Endpunktrichtlinie für HealthImaging

Überlegungen zu HealthImaging VPC-Endpunkten

Bevor Sie einen Schnittstellen-VPC-Endpunkt für einrichten HealthImaging, sollten Sie die Eigenschaften und Einschränkungen der Schnittstellen-Endpunkte im Amazon VPC-Benutzerhandbuch lesen.

HealthImaging unterstützt Aufrufe aller AWS HealthImaging Aktionen von Ihrer VPC aus.

Erstellen eines Schnittstellen-VPC-Endpunkts für HealthImaging

Sie können mithilfe der Amazon VPC-Konsole oder der AWS Command Line Interface ()AWS CLI einen VPC-Endpunkt für den HealthImaging Service erstellen. Weitere Informationen finden Sie unter Erstellung eines Schnittstellenendpunkts im Benutzerhandbuch für Amazon VPC.

Erstellen Sie VPC-Endpoints für die HealthImaging Verwendung der folgenden Dienstnamen:

- com.amazonaws. *region*.medizinische Bildgebung
- com.amazonaws. *region*. runtime-medical-imaging
- com.amazonaws. *region*. dicom-medical-imaging

> **Note**
>
> Private DNS muss für die Verwendung aktiviert sein PrivateLink.

Sie können API-Anfragen an die HealthImaging Verwendung des Standard-DNS-Namens für die Region stellen, `medical-imaging.us-east-1.amazonaws.com` z. B.

Weitere Informationen finden Sie unter Zugriff auf einen Service über einen Schnittstellenendpunkt im Benutzerhandbuch für Amazon VPC.

Erstellen einer VPC-Endpunktrichtlinie für HealthImaging

Sie können eine Endpunktrichtlinie an Ihren VPC-Endpunkt anhängen, der den Zugriff auf HealthImaging steuert. Die Richtlinie gibt die folgenden Informationen an:

- Der Prinzipal, der die Aktionen ausführen kann
- Aktionen, die ausgeführt werden können
- Ressourcen, für die Aktionen ausgeführt werden können

Weitere Informationen finden Sie unter Steuerung des Zugriffs auf Services mit VPC-Endpunkten im Amazon-VPC-Benutzerhandbuch.

Beispiel: VPC-Endpunktrichtlinie für Aktionen HealthImaging

Im Folgenden finden Sie ein Beispiel für eine Endpunktrichtlinie für HealthImaging. Wenn diese Richtlinie an einen Endpunkt angehängt ist, gewährt sie allen Prinzipalen auf allen Ressourcen Zugriff auf HealthImaging Aktionen.

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\":\"*\"}]}"
```

Kontoübergreifender Import für AWS HealthImaging

Mit dem konto-/regionsübergreifenden Import können Sie Daten aus Amazon S3 S3-Buckets in anderen unterstützten Regionen in Ihren HealthImaging Datenspeicher importieren. Sie können Daten zwischen AWS Konten, Konten anderer AWS Organizations und aus offenen Datenquellen wie Imaging Data Commons (IDC) importieren, die sich im Registry of Open Data on AWS befinden.

HealthImaging Zu den Anwendungsfällen für konto-/regionsübergreifende Importe gehören:

- SaaS-Produkte für medizinische Bildgebung importieren DICOM-Daten aus Kundenkonten
- Große Unternehmen, die einen HealthImaging Datenspeicher aus vielen Amazon S3 S3-Eingabe-Buckets befüllen
- Forscher tauschen Daten aus klinischen Studien mit mehreren Institutionen auf sichere Weise aus

Um den kontoübergreifenden Import zu verwenden

1. Der Besitzer des Amazon S3 S3-Eingabe-Buckets (Quell-) Bucket muss dem HealthImaging Datenspeicher-Eigentümer `s3:ListBucket` und `s3:GetObject` Berechtigungen gewähren.
2. Der Besitzer des HealthImaging Datenspeichers muss den Amazon S3 S3-Bucket zu seinem IAM `ImportJobDataAccessRole` hinzufügen. Siehe Erstellen Sie eine IAM-Rolle für den Import.
3. Der Besitzer des HealthImaging Datenspeichers muss den Eingabe-Bucket `inputOwnerAccountId`für den Amazon S3 S3-Eingabe-Bucket angeben, wenn er den Importjob startet.

> **Note**
>
> Durch die Angabe von bestätigt der Besitzer des Datenspeichers`inputOwnerAccountId`, dass der eingegebene Amazon S3 S3-Bucket zu dem angegebenen Konto gehört, um die Einhaltung der Industriestandards zu gewährleisten und potenzielle Sicherheitsrisiken zu minimieren.

Das folgende `startDICOMImportJob` Codebeispiel enthält den optionalen `inputOwnerAccountId` Parameter, der auf alle angewendet werden kann, AWS CLI sowie SDK-Codebeispiele im Einen Importjob starten Abschnitt.

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 "";
}
```

Resilienz in AWS HealthImaging

Die AWS globale Infrastruktur basiert auf Availability AWS-Regionen Zones. AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Falls Sie Ihre Daten oder Anwendungen über größere geografische Distanzen hinweg replizieren müssen, verwenden Sie lokale AWS -Regionen. Eine AWS lokale Region ist ein einzelnes Rechenzentrum, das eine bestehende AWS Region ergänzen soll. Wie alle AWS-Regionen Regionen sind AWS lokale Regionen vollständig von anderen isoliert AWS-Regionen.

Weitere Informationen zu Availability Zones AWS-Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

HealthImaging AWS-Referenzmaterial

> **Note**
>
> Alle nativen HealthImaging API-Aktionen und Datentypen werden in der HealthImaging AWS-API-Referenz beschrieben.

Themen

- DICOM-Unterstützung für AWS HealthImaging
- HealthImaging AWS-Referenz

DICOM-Unterstützung für AWS HealthImaging

AWS HealthImaging unterstützt bestimmte DICOM-Elemente und Übertragungssyntaxen. Machen Sie sich mit den unterstützten DICOM-Datenelementen auf Patienten-, Studien- und Serienebene vertraut, da die HealthImaging Metadatenschlüssel auf diesen basieren. Bevor Sie mit dem Import beginnen, stellen Sie sicher, dass Ihre medizinischen Bilddaten den unterstützten Übertragungssyntaxen und den DICOM-Elementbeschränkungen entsprechen. HealthImaging

> **Note**
>
> AWS unterstützt derzeit HealthImaging keine binären Segmentierungsbilder oder Pixeldaten von Icon Image Sequence.

Themen

- Unterstützte SOP-Klassen
- Normalisierung von Metadaten
- Unterstützte Übertragungssyntaxen
- Einschränkungen für DICOM-Elemente
- Einschränkungen für DICOM-Metadaten

Unterstützte SOP-Klassen

Mit AWS HealthImaging können Sie DICOM P10 Service-Object Pair (SOP) -Instances importieren, die mit einer beliebigen SOP-Klassen-UID codiert sind, einschließlich stillgelegter und privater Instanzen. Alle privaten Attribute werden ebenfalls beibehalten.

Normalisierung von Metadaten

Wenn Sie Ihre DICOM P10-Daten in AWS importieren HealthImaging, werden sie in Bildsätze umgewandelt, die aus Metadaten und Bildrahmen (Pixeldaten) bestehen. Während des Transformationsprozesses werden HealthImaging Metadatenschlüssel auf der Grundlage einer bestimmten Version des DICOM-Standards generiert. HealthImaging generiert und unterstützt derzeit Metadatenschlüssel, die auf dem DICOM PS3 3.6 2022b Data Dictionary basieren.

AWS HealthImaging unterstützt die folgenden DICOM-Datenelemente auf Patienten-, Studien- und Serienebene.

Elemente auf Patientenebene

> **Note**
>
> Eine ausführliche Beschreibung der einzelnen Elemente auf Patientenebene finden Sie im Register der DICOM-Datenelemente.

AWS HealthImaging unterstützt die folgenden Elemente auf Patientenebene:

```
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
```

Elemente auf Studienebene

> **Note**
>
> Eine ausführliche Beschreibung der einzelnen Elemente auf Studienebene finden Sie im Register der DICOM-Datenelemente.

AWS HealthImaging unterstützt die folgenden Elemente auf Studienebene:

```
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
```

Elemente auf Serienebene

> **Note**
>
> Eine ausführliche Beschreibung der einzelnen Elemente auf Serienebene finden Sie in der Registry of DICOM Data Elements.

AWS HealthImaging unterstützt die folgenden Elemente auf Series-Ebene:

```
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
```

Unterstützte Übertragungssyntaxen

AWS HealthImaging unterstützt den Import von DICOM P10-Dateien mit unterschiedlichen Übertragungssyntaxen. Einige Dateien behalten beim Import ihre ursprüngliche Übertragungssyntaxkodierung bei, während die meisten verlustfreien Bildframes während des Imports transkodiert werden. Das Transcodierungsverhalten hängt von Ihrer Datenspeicherkonfiguration ab. Datenspeicher verwenden standardmäßig HTJ2 K als Speicherformat, können aber bei der Erstellung so konfiguriert werden, dass JPEG 2000 Lossless verwendet wird. Das folgende Beispiel zeigt, wie a `StoredTransferSyntaxUID` für jede Instanz innerhalb der von zurückgegebenen Metadaten HealthImaging aufgezeichnet wird `GetImageSetMetadata`.

```
"Instances": {
   "999.999.2.19941105.134500.2.101": {
       "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.90",
       "ImageFrames": [{ ...
```

> **Note**
>
> Beachten Sie diese Punkte, wenn Sie sich die folgende Tabelle ansehen:
>
> - Ein mit einem Sternchen (*) markierter UID-Eintrag in der Übertragungssyntax gibt an, dass eine Datei beim Import in ihrem ursprünglichen codierten Format gespeichert wurde. Bei diesen Dateien entsprechen die in der Instanz `StoredTransferSyntaxUID` befindlichen Metadaten der ursprünglichen Übertragungssyntax.
> - Ein UID-Eintrag in der Übertragungssyntax, der ohne Sternchen gekennzeichnet ist, weist darauf hin, dass eine Datei beim Import verlustfrei in HTJ2 K transkodiert und dort gespeichert wurde. HealthImaging Für das `StoredTransferSyntaxUID` Element der Instanz-Metadaten wird das Speicherformat JPEG 2000 mit hohem Durchsatz und RPCL-Optionen, nur Bildkomprimierung — Lossless (1.2.840.10008.1.2.4.202) festgelegt.

- Wenn der `StoredTransferSyntaxUID` Schlüssel nicht existiert oder auf eingestellt ist, können Sie davon ausgehen, dass er im konfigurierten Datenspeicher-Speicherformat codiert ist. `null`

HealthImaging unterstützte Übertragungssyntaxen

Übertragungssyntax UID	Name der Übertragungssyntax
1.2.840.10008.1.2	Implizites VR Endian: Standard-Übertragungssyntax für DICOM
1.2.840.10008.1.2.1* (Die binäre Segmentierung behält die ursprüngliche Kodierung bei, wohingegen die nichtbinäre Segmentierung in K Lossless RPCL transkodiert wird) HTJ2	Expliziter VR Little Endian
1.2.840.10008.1.2.1.99	Deflationierter Expliziter VR Little Endian
1.2.840.10008.1.2.2	Expliziter VR Big Endian
1.2.840.10008.1.2.4.50*	JPEG-Baseline (Prozess 1): Standardübertragungssyntax für die verlustbehaftete 8-Bit-JPEG-Bildkomprimierung
1.2.840.10008.1.2.4.51	JPEG-Baseline (Prozesse 2 und 4): Standardübertragungssyntax für die verlustbehaftete 12-Bit-JPEG-Bildkomprimierung (nur Prozess 4)
1.2.840.10008.1.2.4.57	JPEG Lossless, nicht hierarchisch (Prozess 14)
1.2.840.10008.1.2.4.70	Verlustfreies JPEG, nicht hierarchisch, Vorhersage erster Ordnung (Prozesse 14 [Auswahlwert 1]): Standardübertragungssyntax für die verlustfreie JPEG-Bildkomprimierung
1.2.840.10008.1.2.4.80	JPEG-LS Verlustfreie Bildkomprimierung

Übertragungssyntax UID	Name der Übertragungssyntax
1.2.840.10008.1.2.4.81	JPEG-LS Verlustbehaftete (nahezu verlustfreie) Bildkomprimierung
1.2.840.10008.1.2.4.90	JPEG 2000-Bildkomprimierung (nur verlustfrei)
1.2.840.10008.1.2.4.91*	JPEG 2000-Bildkomprimierung
1.2.840.10008.1.2.4.92	JPEG 2000 Teil 2 Mehrkomponenten-Bi ldkomprimierung (nur verlustfrei)
1.2.840.10008.1.2.4.93	JPEG 2000 Teil 2 Mehrkomponenten-Bi ldkomprimierung
1.2.840.10008.1.2.4.112*	JPEG XL
1.2.840.10008.1.2.4.201	JPEG 2000-Bildkomprimierung mit hohem Durchsatz (nur verlustfrei)
1.2.840.10008.1.2.4.202	JPEG 2000 mit hohem Durchsatz und Bildkomprimierung mit RPCL-Optionen (nur verlustfrei)
1.2.840.10008.1.2.4.203*	JPEG 2000-Bildkomprimierung mit hohem Durchsatz
1.2.840.10008.1.2.5	RLE Verlustfrei
1.2.840.10008.1.2.4.100*, 1.2.840.10008.1.2. 4.100.1*	MPEG2 Hauptprofil Hauptebene
1.2.840.10008.1.2.4.101*, 1.2.840.10008.1.2. 4.101.1*	MPEG2 Hauptprofil auf hohem Niveau
1.2.840.10008.1.2.4.102*, 1.2.840.10008.1.2. 4.102.1*	MPEG-4 AVC/H.264 mit hohem Bekannthe itsgrad//Stufe 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-kompatibles High Profile/Level 4.1

Übertragungssyntax UID	Name der Übertragungssyntax
1.2.840.10008.1.2.4.104*, 1.2.840.10008.1.2.4.104.1*	MPEG-4 AVC/H.264 High Profile//Stufe 4.2 für 2D-Video
1.2.840.10008.1.2.4.105*, 1.2.840.10008.1.2.4.105,1*	MPEG-4 AVC/H.264 High Profile/Stufe 4.2 des ITU-T H.264-Videos
1.2.840.10008.1.2.4.106*, 1.2.840.10008.1.2.4.106,1*	MPEG-4 AVC/H.264 Stereo High Profile/Stufe 4.2 des ITU-T H.264-Videos
1.2.840.10008.1.2.4.107*	HEVC/H.265-Hauptprofil/Level 5.1-Video
1.2.840.10008.1.2.4.108*	HEVC/H.265-Hauptprofil 10//Stufe 5.1

*Behält die ursprüngliche Kodierung der Übertragungssyntax beim Import bei

Einschränkungen für DICOM-Elemente

Beim Import Ihrer medizinischen Bilddaten in AWS HealthImaging gelten maximale Längenbeschränkungen für die folgenden DICOM-Elemente. Um einen erfolgreichen Import zu erreichen, stellen Sie sicher, dass Ihre Daten die maximale Länge nicht überschreiten.

Einschränkungen für DICOM-Elemente beim Import

DICOM-Schlüsselwort	DICOM-Tag	Max. Länge
PatientName	(0010.0010)	256
ID des Patienten	(0010,0020)	256
PatientBirthDate	(0010.0030)	18
PatientSex	(0010.0040)	16
StudyInstanceUID	(0020.000 D)	256
Studien-ID	(0020,0010)	256
StudyDescription	(0008.1030)	256

DICOM-Schlüsselwort	DICOM-Tag	Max. Länge
NumberOfStudyRelatedSeries	(0020.1206)	1 000 000
NumberOfStudyRelatedInstances	(0020.1208)	1 000 000
AccessionNumber	(0008.0050)	256
StudyDate	(0008.0020)	18
StudyTime	(0008.0030)	28
SOPInstanceUID	(0008.0018)	256
SeriesInstanceUID	(0020.000 E)	256

Einschränkungen für DICOM-Metadaten

Bei der Aktualisierung von `UpdateImageSetMetadata` HealthImaging Metadatenattributen werden die folgenden DICOM-Einschränkungen angewendet.

- Private Attribute auf Patient/Study/Series/Instance Ebenenattributen können nur aktualisiert oder entfernt werden, wenn die Aktualisierungsbeschränkung sowohl für als auch `updatableAttributes` gilt `removableAttributes`
- Die folgenden von AWS HealthImaging generierten Attribute können nicht aktualisiert werden: `SchemaVersion` `DatastoreID` `ImageSetIDPixelData`,`Checksum`,`Width`,`Height`,`MinPixelValue`,`MaxPixelValue`, `FrameSizeInBytes`
- Die folgenden DICOM-Attribute können nur aktualisiert werden, wenn das `force` Flag gesetzt ist:`Tag.PixelData`,`Tag.StudyInstanceUID`,`Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID` `Tag.StudyID`
- Attribute mit VR-Typ SQ (verschachtelte Attribute) können nur aktualisiert werden, wenn das `force` Flag gesetzt ist
- Mehrwertige Attribute können nur aktualisiert werden, wenn das `force` Flag gesetzt ist
- Attribute mit Werten, die nicht mit dem Attribut-VR-Typ kompatibel sind, können nur aktualisiert werden, wenn das `force` Flag gesetzt ist

- Attribute, die gemäß dem DICOM-Standard nicht als gültige Attribute gelten, können nur aktualisiert werden, wenn das `force` Flag gesetzt ist
- Attribute können nicht modulübergreifend aktualisiert werden. Wenn beispielsweise in der Payload-Anfrage des Kunden auf Studienebene ein Attribut auf Patientenebene angegeben wird, kann die Anfrage für ungültig erklärt werden.
- Attribute können nicht aktualisiert werden, wenn das zugehörige Attributmodul nicht vorhanden ist. `ImageSetMetadata` Sie dürfen beispielsweise keine Attribute für a aktualisieren, `seriesInstanceUID` wenn die Serie mit nicht in den vorhandenen Bilddatensatz-Metadaten enthalten `seriesInstanceUID` ist.

HealthImaging AWS-Referenz

Dieser Abschnitt enthält unterstützende Daten, die für AWS relevant sind HealthImaging.

Themen

- HealthImaging AWS-Endpunkte und Kontingente
- HealthImaging AWS-Drosselungsgrenzen
- Überprüfung AWS HealthImaging AWS-Pixeldaten
- HealthImaging Warncodes
- Bibliotheken zur Dekodierung von Image-Frames für AWS HealthImaging
- HealthImaging AWS-Beispielprojekte
- Verwenden Sie diesen Service mit einem AWS SDK

HealthImaging AWS-Endpunkte und Kontingente

Die folgenden Themen enthalten Informationen zu HealthImaging AWS-Servicendpunkten und -kontingenten.

Themen

- Service-Endpunkte
- Servicekontingente

Service-Endpunkte

Ein Service-Endpunkt ist eine URL, die einen Host und einen Port als Einstiegspunkt für einen Webservice identifiziert. Jede Webserviceanforderung umfasst einen Endpunkt. Die meisten AWS Dienste bieten Endpunkte für bestimmte Regionen, um eine schnellere Konnektivität zu ermöglichen. In der folgenden Tabelle sind die Service-Endpunkte für AWS HealthImaging aufgeführt.

Name der Region	Region	Endpunkt	Protocol (Protokoll)
USA Ost (Nord-Virginia)	us-east-1	medical-imaging.us-east-1.amazonaws.com medical-imaging-fips.us-east-1.amazonaws.com	HTTPS HTTPS
USA West (Oregon)	us-west-2	medical-imaging.us-west-2.amazonaws.com medical-imaging-fips.us-west-2.amazonaws.com	HTTPS HTTPS
Asien-Pazifik (Sydney)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS
Europa (Irland)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS
Europa (London)	eu-west-2	medical-imaging.eu-west-2.amazonaws.com	HTTPS

Wenn Sie HTTP-Anfragen zum Aufrufen von HealthImaging AWS-Aktionen verwenden, müssen Sie je nach aufgerufenen Aktionen unterschiedliche Endpunkte verwenden. Das folgende Menü listet die verfügbaren Service-Endpunkte für HTTP-Anfragen und die von ihnen unterstützten Aktionen auf.

Unterstützte API-Aktionen für HTTP-Anfragen

data store, import, tagging

Auf die folgenden Datenspeicher -, Import - und Tagging-Aktionen kann über den Endpunkt zugegriffen werden:

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- StartDICOMImportJob
- GetDICOMImportJob
- ListDICOMImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

Auf die folgenden Bilddatensatz-Aktionen kann über den Endpunkt zugegriffen werden:

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging bietet eine Darstellung der DICOMWeb Retrieve-Dienste. WADO-RS Weitere Informationen finden Sie unter DICOM-Daten werden abgerufen von HealthImaging.

Auf die folgenden DICOMWeb-Dienste kann über einen Endpunkt zugegriffen werden:

```
https://dicom-medical-imaging.region.amazonaws.com
```

- `GetDICOMInstance`
- `GetDICOMInstanceMetadata`
- `GetDICOMInstanceFrames`

Servicekontingente

Dienstkontingente sind als der Höchstwert für Ihre Ressourcen, Aktionen und Elemente in Ihrem AWS Konto definiert.

> **Note**
>
> Für anpassbare Kontingente können Sie über die Service Quotas-Konsole eine Erhöhung des Kontingents beantragen. Weitere Informationen finden Sie unter Beantragen einer Kontingenterhöhung im Service-Quotas-Benutzerhandbuch.

In der folgenden Tabelle sind die Standardkontingente für AWS aufgeführt HealthImaging.

Name	Standard	Anpas	Description
Maximale Anzahl gleichzeitiger CopyImageSet Anfragen pro Datenspeicher	Jede unterstützte Region: 100	Yes (Ja)	Die maximale Anzahl gleichzeitiger CopyImageSet Anfragen pro Datenspeicher in der aktuellen Region AWS
Maximale Anzahl gleichzeitiger DeleteImageSet Anfragen pro Datenspeicher	Jede unterstützte Region: 100	Yes (Ja)	Die maximale Anzahl gleichzeitiger DeleteImageSet Anfragen pro Datenspeicher in der aktuellen Region AWS
Maximale Anzahl gleichzeitiger UpdateImageSetMetadata Anfragen pro Datenspeicher	Jede unterstützte Region: 100	Yes (Ja)	Die maximale Anzahl gleichzeitiger UpdateImageSetMetadata Anfragen pro Datenspeicher in der aktuellen Region AWS
Maximale Anzahl gleichzeitiger Importaufträge pro Datenspeicher	ap-southeast-2:20 Jede der anderen unterstützten Regionen: 100	Ja	Die maximale Anzahl gleichzeitiger Importaufträge pro Datenspeicher in der aktuellen Region AWS
Maximale Anzahl von Datenspeichern	Jede unterstützte Region: 10	Yes (Ja)	Die maximale Anzahl aktiver Datenspeicher

Name	Standard	Anpas	Description
			in der aktuellen AWS Region
Maximale Anzahl von Dateien, die pro CopyImageSet Anfrage kopiert werden ImageFrames dürfen	Jede unterstützte Region: 1 000	Ja	Die maximale Anzahl von Kopien, die pro CopyImageSet Anfrage in der aktuellen AWS Region kopiert werden ImageFrames dürfen
Maximale Anzahl von Dateien in einem DICOM-Importauftrag	Jede unterstützte Region: 5 000	Ja	Die maximale Anzahl von Dateien in einem DICOM-Importauftrag in der aktuellen Region AWS
Maximale Anzahl verschachtelter Ordner in einem DICOM-Importauftrag	Jede unterstützte Region: 10 000	Nein	Die maximale Anzahl verschachtelter Ordner in einem DICOM-Imp ortauftrag in der aktuellen Region AWS
Die maximale Größenbeschränkung für Payloads (in KB) wurde akzeptiert von UpdateImageSetMetadata	Jede unterstüt zte Region: 10 Kilobyte	Ja	Die maximale Nutzlastg rößenbeschränkung (in KB), die von der aktuellen UpdateImageSetMeta data Region akzeptiert wird AWS
Maximale Größe (in GB) aller Dateien in einem DICOM-Importauftrag	Jede unterstüt zte Region: 10 Gigabyte	Nein	Die maximale Größe (in GB) aller Dateien in einem DICOM-Imp ortauftrag in der aktuellen Region AWS

Name	Standard	Anpas	Description
Maximale Größe (in GB) jeder DICOM P10-Datei in einem DICOM-Importauftrag	Jede unterstützte Region: 4 Gigabyte	Nein	Die maximale Größe (in GB) jeder DICOM P10-Datei im DICOM-Importauftrag in der aktuellen Region AWS
Maximale Größenbeschränkung (in MB) ImageSetMetadata pro Import, Kopie und UpdateImageSet	Jede unterstützte Region: 50 Megabyte	Ja	Die maximale Größenbeschränkung (in MB) ImageSetMetadata pro Import, Kopie und UpdateImageSet in der aktuellen AWS Region

HealthImaging AWS-Drosselungsgrenzen

Für Ihr AWS Konto gelten Drosselungslimits, die für HealthImaging AWS-API-Aktionen gelten. Bei allen Aktionen wird ein `ThrottlingException` Fehler ausgegeben, wenn die Drosselungsgrenzen überschritten werden. Weitere Informationen finden Sie in der HealthImaging AWS-API-Referenz.

> **Note**
>
> Die Drosselungsgrenzen sind für alle HealthImaging API-Aktionen einstellbar. Um eine Anpassung der Drosselungsgrenze zu beantragen, wenden Sie sich an das AWS Support Center. Um einen Fall zu erstellen, melden Sie sich bei Ihrem AWS Konto an und wählen Sie Kundenvorgang erstellen.

In der folgenden Tabelle sind die Drosselungsgrenzen sowohl für native HealthImaging Aktionen als auch für Repräsentationen von DICOMweb Diensten aufgeführt.

HealthImaging AWS-Drosselungsgrenzen

Action	Drosselungsrate	Gaspedal geplatzt
CreateDatastore	0,085 Tips	1 TL

Action	Drosselungsrate	Gaspedal geplatzt
GetDatastore	10 TPS	20 Teelöffel
ListDatastores	5 Teelöffel	10 TPS
DeleteDatastore	0,085 TL	1 TL
DICOMImportJob starten	10 Tipps	2 Teelöffel
DICOMImportJob bekommen	25 Tipps	50 Teelöffel
Jobs auflisten DICOMImport	10 TPS	20 Tipps
SearchImageSets	25 Teelöffel	50 Teelöffel
GetImageSet	25 Teelöffel	50 Teelöffel
GetImageSetMetadata	50 Teelöffel	100 Teelöffel
GetImageFrame	1000 Teelöffel	2000 tps
ListImageSetVersions	25 Teelöffel	50 Teelöffel
UpdateImageSetMetadata	0,25 Teelöffel	10 Teelöffel
CopyImageSet	0,25 Teelöffel	10 Teelöffel
DeleteImageSet	0,25 Teelöffel	10 Teelöffel
TagResource	10 TPS	20 Teelöffel
ListTagsForResource	10 TPS	20 Teelöffel
UntagResource	10 TPS	20 Teelöffel
Holen Sie sich * DICOMInstance	50 Tipps	100 Teelöffel
Metadaten abrufen* DICOMInstance	50 Tipps	100 Teelöffel

Action	Drosselungsrate	Gaspedal geplatzt
Holen Sie sich Frames* DICOMInstance	50 Tipps	100 Teelöffel
Metadaten abrufen DICOMSeries	50 Tipps	100 Teelöffel

*Darstellung einer Dienstleistung DICOMweb

Überprüfung AWS HealthImaging AWS-Pixeldaten

HealthImaging Bietet während des Imports eine integrierte Überprüfung der Pixeldaten, indem der Status der verlustfreien Kodierung und Dekodierung jedes Bilds überprüft wird. Diese Funktion stellt sicher, dass Bilder, die mit HTJ2K-Dekodierungsbibliotheken dekodiert wurden, immer mit den ursprünglichen DICOM P10-Bildern übereinstimmen, in die importiert wurde. HealthImaging

- Der Bild-Onboarding-Prozess beginnt, wenn ein Importauftrag den ursprünglichen Pixelqualitätsstatus von DICOM P10-Bildern erfasst, bevor sie importiert werden. Mithilfe des Algorithmus wird für jedes Bild eine eindeutige, unveränderliche Prüfsumme für die Bildauflösung (IFRC) generiert. CRC32 Der IFRC-Prüfsummenwert wird im Metadatendokument dargestellt. `job-output-manifest.json` Weitere Informationen finden Sie unter Importaufträge verstehen.
- Nachdem die Bilder in einen HealthImaging Datenspeicher importiert und in Bilddatensätze umgewandelt wurden, werden die HTJ2 K-kodierten Bildrahmen sofort dekodiert und neue werden berechnet. IFRCs HealthImaging vergleicht dann die volle Auflösung IFRCs der Originalbilder mit den neuen IFRCs importierten Bildrahmen, um die Genauigkeit zu überprüfen.
- Ein entsprechender beschreibender Fehler pro Bild wird im Ausgabeprotokoll des Importauftrags (`job-output-manifest.json`) erfasst, damit Sie ihn überprüfen und verifizieren können.

Um Pixeldaten zu verifizieren

1. Sehen Sie sich nach dem Import Ihrer medizinischen Bilddaten die Beschreibung des Erfolgs (oder Fehlerzustands) pro Bilddatensatz an, die im Ausgabeprotokoll des Importauftrags erfasst wurde. `job-output-manifest.json` Weitere Informationen finden Sie unter Importaufträge verstehen.

2. Bilddatensätze bestehen aus Metadaten und Bildrahmen (Pixeldaten). Bildsatz-Metadaten enthalten Informationen über zugehörige Bildrahmen. Verwenden Sie die `GetImageSetMetadata` Aktion, um Metadaten für einen Bildsatz abzurufen. Weitere Informationen finden Sie unter Metadaten von Bilddatensätzen abrufen.

3. Das `PixelDataChecksumFromBaseToFullResolution` enthält die IFRC (Prüfsumme) für das Bild mit voller Auflösung. Für Bilder, die in ihrer ursprünglichen Übertragungssyntax 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 und 1.2.840.10008.1.2.1 (nur binäre Segmentierung) gespeichert wurden, wird die Prüfsumme anhand des Originalbilds berechnet. Für Bilder, die HTJ2 in K Lossless mit RPCL gespeichert sind, wird die Prüfsumme anhand des dekodierten Bildes mit voller Auflösung berechnet. Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen.

   Im Folgenden finden Sie ein Beispiel für die Metadatenausgabe für die IFRC, die im Rahmen des Importauftrags generiert und aufgezeichnet wird. `job-output-manifest.json`

   ```
   "ImageFrames": [{
   "ID": "678906789067890123451234512",
   "PixelDataChecksumFromBaseToFullResolution": [
   {
       "Width": 512,
       "Height": 512,
       "Checksum": 2510355201
   }
   ]
   ```

   Für Bilder, die in ihrer ursprünglichen Übertragungssyntax 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 und 1.2.840.10008.1.2.1 (nur binäre Segmentierung) gespeichert wurden, ist das und nicht verfügbar. `MinPixelValue` `MaxPixelValue` `FrameSizeInBytes`Das gibt die Größe des Originalrahmens an.

   ```
   "PixelDataChecksumFromBaseToFullResolution": [
      {"Width": 512, "Height": 512, "Checksum": 1379921327 }
   ],
   "MinPixelValue": null,
   "MaxPixelValue": null,
   "FrameSizeInBytes": 429
   ```

   Bei Bildern, die in HTJ2 K Lossless mit RPCL gespeichert wurden, `FrameSizeInBytes` gibt das die Größe des dekodierten Bildrahmens an.

```
"PixelDataChecksumFromBaseToFullResolution": [
   {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": 11,
"MaxPixelValue": 11,
"FrameSizeInBytes": 1652
```

4. HealthImaging Führt bei Instances, die Video enthalten, eine Lightweight-Codec-Validierung durch, bei der überprüft wird, ob die in den DICOM-Metadaten angegebene Übertragungssyntax mit dem Videocodec übereinstimmt.

   HealthImaging berechnet mithilfe des Algorithmus einen IFRC-Prüfsummenwert für das ursprüngliche Videoobjekt. CRC32 Der IFRC-Prüfsummenwert wird in den Metadaten aufgezeichnet und in den `job-output-manifest.json` Metadaten gespeichert. HealthImaging Wie bei Bildern, die in ihrer ursprünglichen Übertragungssyntax (oben beschrieben) gespeichert wurden, ist `MaxPixelValue` das `MinPixelValue` und nicht verfügbar. Das FrameSizeInBytes gibt die Größe des Originalrahmens an.

5. HealthImaging verifizieren Sie die Pixeldaten, rufen Sie das Verfahren zur Überprüfung der Pixeldaten auf GitHub und folgen Sie den Anweisungen in der `README.md` Datei, um die verlustfreie Bildverarbeitung durch Bibliotheken zur Dekodierung von Bildrahmen die verschiedenen, von verwendeten Geräten unabhängig zu überprüfen HealthImaging. Nachdem das vollständige Bild geladen wurde, können Sie den IFRC für Rohdaten selbst berechnen und ihn mit dem in den HealthImaging Metadaten angegebenen IFRC-Wert vergleichen, um die Pixeldaten zu verifizieren.

HealthImaging Warncodes

HealthImaging versucht, all Ihre medizinischen Bilddaten zu importieren. Wenn bei Importen Datenabweichungen oder unbekannte Datenelemente auftreten, HealthImaging wird der Datei eine der folgenden Warnungen hinzugefügt. `warning.ndjson` Eine Warnung, die mit einer importierten Instanz verknüpft ist, kann auch über die Aktion mit dem `SearchDICOMInstances` Element durchsucht werden. `WarningReason` Für Instanzen, die mit einer Warnung importiert wurden, wurde der Support von via möglicherweise eingeschränkt HealthImaging APIs, wie unten beschrieben.

HealthImaging Warncodes importieren

Grund für die Warnung (Hexadezimal)	Grund der Warnung (dezimal)	Art der Warnung (Enum)	Einzelheiten zur Warnung	Resultierendes Verhalten
		Gründe für DICOM-Standardwarnungen		
0xB000	45056	ZWANG VON_DATEN ELEMENTEN	Durch die Aufnahme wurden ein oder mehrere Datenelemente beim Speichern der Instanz geändert. Siehe Abschnitt 6.6.1.3.	–
0xB006	45062	ELEMENTE_ VERWORFE	Bei der Aufnahme wurden einige Datenelemente beim Speichern der Instanz verworfen. Siehe Abschnitt 6.6.1.3.	–
0xB007	45063	SOP_CLASS _DATA_MIS MATCH	Bei der `StoreDICOM` Aktion wurde festgestellt, dass der Datensatz beim Speichern der Instanz nicht den Einschrän kungen der SOP-Klasse entsprach.	–
		HealthImaging AWS-Warngründe		
0xB100	45312	TRANSCODI ERUNG_AUS NAHME	Diese Warnung tritt auf, wenn die Instanz HealthImaging PixelData nicht in HTJ2 K (Standard speicherformat) transkodi ert werden kann oder	Pixeldaten können bei Bedarf weiterhin als einzelner Blob abgerufen werden. Wenn Sie den Platzhalter „*“ als In-Accept-Header

Grund für die Warnung (Hexadezimal)	Grund der Warnung (dezimal)	Art der Warnung (Enum)	Einzelheiten zur Warnung	Resultierendes Verhalten
			wenn sie aus einem anderen Grund nicht transkodiert werden PixelData kann (d. h. fehlgeschlagene Validierungen, falsche Pixeldatenattribute usw.). In diesem Fall werden die Pixeldaten als Blob gespeichert.	übergeben, werden sie `transfer-syntax` im gespeicherten Format zurückgegeben.
0xB110	45328	FEHLER BEIM EXTRAHIEREN VON FRAMES	Diese Warnung tritt auf, wenn beim Parsen einzelner Frames anhand der angegebenen DICOM-Metadaten ein Problem auftritt. PixelData	Die Pixeldaten waren falsch formatiert und können nicht abgerufen werden. Verwenden Sie diese Option, `GetDICOMInstance` um die gesamte Instanz abzurufen
0xB111	45329	FRAME_NUMBER_STIMMT NICHT ÜBEREIN	Diese Warnung tritt auf, wenn das "NumberOfFrames" DICOM-Element nicht der tatsächlichen Anzahl von Bild-„Fragmenten“ in der DICOM-Eingabedatei entspricht.	Die Pixeldaten waren falsch formatiert und können nicht abgerufen werden. Verwenden Sie diese Option, `GetDICOMInstance` um die gesamte Instanz abzurufen

Grund für die Warnung (Hexadezimal)	Grund der Warnung (dezimal)	Art der Warnung (Enum)	Einzelheiten zur Warnung	Resultierendes Verhalten
0xB112	45330	UNGÜLTIGE_OFFSET-TABELLE	Diese Warnung tritt auf, wenn die Offset-Tabelle in den Fragmenten der DICOM-Eingabedatei nicht mit der tatsächlichen Framelänge übereinstimmt und je nach Schweregrad zu falsch formatierten Frames führen kann.	Die Pixeldaten waren falsch formatiert und können nicht abgerufen werden. Verwenden Sie diese Option, `GetDICOMInstance` um die gesamte Instanz abzurufen
0xB120	45344	NICHT UNTERSTÜTZTE_TRANSFER-SYNTAX	Diese Warnung tritt auf, wenn auf eine unerkannte oder nicht unterstützte HealthImaging Übertragungssyntax gestoßen wird. In diesem Fall HealthImaging werden die Pixeldaten als Blob gespeichert.	Pixeldaten können bei Bedarf weiterhin als einzelner Blob abgerufen werden. Wenn Sie den Platzhalter „*“ als In-Accept-Header übergeben, werden sie `transfer-syntax` im gespeicherten Format zurückgegeben.
0xB201	45570	UNGÜLTIGES UID-FORMAT	Diese Warnung tritt auf, wenn ein oder mehrere UID-Elemente gegen die DICOM-Wertdarstellung verstoßen (z. B.) `1.2.3..4`	–

Grund für die Warnung (Hexadezimal)	Grund der Warnung (dezimal)	Art der Warnung (Enum)	Einzelheiten zur Warnung	Resultierendes Verhalten
0xB202	45571	UNGÜLTIGE_DICOM-WERTELÄNGE	Diese Warnung tritt auf, wenn ein DICOM-Element eine Länge hat, die länger ist als die von der DICOM-Wertrepräsentation unterstützte Länge, was möglicherweise zu ungültigen Verhaltensweisen für Aktionen führen kann. search/retrieve	Einige Felder sind möglicherweise nicht analysierbar und können daher nicht nach ihnen gesucht werden (d. h. oder) `StudyDate` `StudyTime`
0xBFFF	47513	OTHER	Diese Warnung tritt auf, wenn es eine nicht abgefangene Warnung gibt, die HealthImaging nicht als spezifischer Warncode erfasst wird.	Die Pixeldaten waren falsch formatiert und können nicht abgerufen werden. Verwenden Sie diese Option, `GetDICOMInstance` um die gesamte Instanz abzurufen

Bibliotheken zur Dekodierung von Image-Frames für AWS HealthImaging

Während des Imports behalten einige Übertragungssyntaxen ihre ursprüngliche Kodierung bei, während andere je nach Ihrer Datenspeicherkonfiguration standardmäßig in verlustfreies High-Throughput-JPEG 2000 (HTJ2K) oder JPEG 2000 Lossless (sofern konfiguriert) transkodiert werden. HTJ2K bietet eine gleichbleibend schnelle Bildanzeige und bietet universellen Zugriff auf die erweiterten Funktionen von HTJ2 K. Da Bildrahmen beim Import entweder in HTJ2 K oder JPEG 2000 Lossless codiert werden, müssen sie vor der Anzeige in einem Bildbetrachter dekodiert werden. Informationen zur Bestimmung der Übertragungssyntaxen finden Sie unter Unterstützte

Übertragungssyntaxen. Hinweise zur Bestimmung der Übertragungssyntaxen finden Sie unter. Unterstützte Übertragungssyntaxen

> **Note**
>
> HTJ2K ist in Teil 15 der Norm JPEG2 000 (ISO/IEC 15444-15:2019) definiert. HTJ2K behält die erweiterten Funktionen von JPEG2 000 wie Skalierbarkeit der Auflösung, Bezirke, Kachelung, hohe Bittiefe, mehrere Kanäle und Farbraumunterstützung bei.

Themen

- Bibliotheken zur Dekodierung von Bildrahmen
- Bildbetrachter

Bibliotheken zur Dekodierung von Bildrahmen

Abhängig von Ihrer Programmiersprache empfehlen wir die folgenden Dekodierungsbibliotheken, um Bildrahmen zu dekodieren.

- NVIDIA nv JPEG2 000 — Kommerziell, GPU-beschleunigt
- Kakadu Software — Kommerziell, C++ mit Java- und .NET-Bindungen
- OpenJPH — Open Source, C++ und WASM
- OpenJPEG — Open Source, C/C++, Java
- openjphpy — Open Source, Python
- pylibjpeg-openjpeg — Open Source, Python

Bildbetrachter

Sie können Bildrahmen ansehen, nachdem Sie sie dekodiert haben. HealthImaging AWS-API-Aktionen unterstützen eine Vielzahl von Open-Source-Bildbetrachtern, darunter:

- Open Health Imaging Foundation (OHIF)
- Cornerstone.js

HealthImaging AWS-Beispielprojekte

AWS HealthImaging bietet die folgenden Beispielprojekte für GitHub.

OHIF Viewer ist HealthImaging über OIDC in AWS integriert

Dieses AWS Cloud Development Kit (AWS CDK)Projekt stellt den OHIF Viewer auf Amazon bereit. CloudFront Der Viewer ist in einen Amazon Web Services Services-Datenspeicher als DICOMWeb Datenquelle und mit Amazon Cognito als Identitätsanbieter für die Authentifizierung über OIDC integriert.

DICOM-Erfassung von On-Premise zu AWS HealthImaging

Ein AWS serverloses Projekt zur Bereitstellung einer IoT-Edge-Lösung, die DICOM-Dateien von einer DICOM-DIMSE-Quelle (PACS, VNA, CT-Scanner) empfängt und sie in einem sicheren Amazon S3 S3-Bucket speichert. Die Lösung indexiert die DICOM-Dateien in einer Datenbank und stellt jede DICOM-Serie, die in AWS importiert werden soll, in eine Warteschlange. HealthImaging Sie besteht aus einer Komponente, die am Edge läuft und von der Cloud verwaltet wird, und einer AWS IoT GreengrassDICOM-Erfassungspipeline, die in der Cloud läuft. AWS

TLM-Proxy (Tile Level Marker)

Ein AWS Cloud Development Kit (AWS CDK)Projekt zum Abrufen von Bildrahmen aus AWS mithilfe HealthImaging von Tile Level Markers (TLM), einer Funktion von High-Throughput JPEG 2000 (HTJ2K). Dies führt zu schnelleren Abrufzeiten bei Bildern mit niedrigerer Auflösung. Zu den möglichen Arbeitsabläufen gehören das Generieren von Miniaturansichten und das schrittweise Laden von Bildern.

CloudFront Amazon-Lieferung

Ein AWS serverloses Projekt zur Erstellung einer CloudFrontAmazon-Distribution mit einem HTTPS-Endpunkt, der Image-Frames vom Edge zwischenspeichert (mithilfe von GET) und bereitstellt. Standardmäßig authentifiziert der Endpunkt Anfragen mit einem Amazon Cognito JSON Web Token (JWT). Sowohl die Authentifizierung als auch das Signieren von Anfragen werden am Edge mit Lambda @Edge durchgeführt. Dieser Service ist eine Funktion von Amazon CloudFront , mit der Sie Code näher an den Benutzern Ihrer Anwendung ausführen können, wodurch die Leistung verbessert und die Latenz reduziert wird. Es muss keine Infrastruktur verwaltet werden.

HealthImaging AWS-Viewer-Benutzeroberfläche

Ein AWS AmplifyProjekt zur Bereitstellung einer Frontend-Benutzeroberfläche mit Backend-Authentifizierung, mit der Sie Metadatenattribute und Bildrahmen (Pixeldaten), die in AWS gespeichert sind, HealthImaging mithilfe progressiver Dekodierung anzeigen können. Sie können optional die oben genannten Tile Level Marker (TLM) Proxy-Projekte von and/or Amazon CloudFront Delivery integrieren, um Bildrahmen mit einer alternativen Methode zu laden.

HealthImaging DICOMweb AWS-Proxy

Ein Python-basiertes Projekt zur Aktivierung von DICOMweb WADO-RS- und QIDO-RS-Endpunkten in einem HealthImaging Datenspeicher zur Unterstützung von webbasierten Viewern für medizinische Bildgebung und anderen kompatiblen Apps. DICOMweb

> **Note**
>
> Dieses Projekt verwendet nicht die unter beschriebene Darstellung. HealthImaging DICOMweb APIs Verwendung DICOMweb mit AWS HealthImaging

Weitere Beispielprojekte finden Sie unter HealthImaging AWS-Beispiele auf GitHub.

Verwenden Sie diesen Service mit einem AWS SDK

AWS Software Development Kits (SDKs) sind für viele beliebte Programmiersprachen verfügbar. Jedes SDK bietet eine API, Codebeispiele und Dokumentation, die es Entwicklern erleichtern, Anwendungen in ihrer bevorzugten Sprache zu erstellen.

SDK-Dokumentation	Codebeispiele
AWS SDK für C++	AWS SDK für C++ Codebeispiele
AWS CLI	AWS CLI Code-Beispiele
AWS SDK für Go	AWS SDK für Go Code-Beispiele
AWS SDK für Java	AWS SDK für Java Code-Beispiele
AWS SDK für JavaScript	AWS SDK für JavaScript Code-Beispiele
AWS SDK für Kotlin	AWS SDK für Kotlin Code-Beispiele

SDK-Dokumentation	Codebeispiele
AWS SDK für .NET	AWS SDK für .NET Code-Beispiele
AWS SDK für PHP	AWS SDK für PHP Code-Beispiele
AWS -Tools für PowerShell	AWS -Tools für PowerShell Code-Beispiele
AWS SDK für Python (Boto3)	AWS SDK für Python (Boto3) Code-Beispiele
AWS SDK für Ruby	AWS SDK für Ruby Code-Beispiele
AWS SDK für Rust	AWS SDK für Rust Code-Beispiele
AWS SDK für SAP ABAP	AWS SDK für SAP ABAP Code-Beispiele
AWS SDK für Swift	AWS SDK für Swift Code-Beispiele

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie ein Codebeispiel an, indem Sie unten den Link Provide feedback (Feedback geben) auswählen.

Kostenoptimierung

HealthImaging wurde entwickelt, um die Speicherkosten zu optimieren, indem Daten automatisch auf die kostengünstigste Zugriffsebene verschoben werden, wenn sich die Zugriffsmuster ändern. Da sich die Nutzungsmuster im Laufe der Zeit ändern, ermöglicht intelligentes Tiering ein automatisches Lebenszyklusmanagement für DICOM-Daten ohne jeglichen betrieblichen Aufwand. HealthImaging hat zwei Speicherstufen:

- Speicher auf der Stufe „Frequent Access“ für neu importierte oder regelmäßig abgerufene DICOM-Daten.
- Archivieren Sie den Instant-Access-Tier-Speicher für DICOM-Daten, auf die in letzter Zeit nicht zugegriffen wurde. Bietet geringere Speicherkosten für die Langzeitarchivierung bei gleichzeitiger Beibehaltung von Zugriffslatenzen im Millisekundenbereich.

Ihnen wird die Gesamtspeichergröße aller Bildsätze in einem Datenspeicher in Rechnung gestellt. Beide Speicherstufen werden pro GB pro Monat abgerechnet, und es fällt keine Gebühr pro Image-Set an. Es fallen auch keine Abrufgebühren für das Verschieben von Daten zwischen Speicherstufen an.

> **Note**
>
> Bilddatensätze werden mit einer Mindestgröße von 5 MB abgerechnet. HealthImaging akzeptiert Bilddatensätze, die kleiner als 5 MB sind, aber für diese kleineren Objekte wird ein Tarif von 5 MB berechnet.

So funktioniert Intelligentes Tiering

Ihre Daten werden in der Stufe „Häufiger Zugriff“ gespeichert, wenn neue Bilddatensätze erstellt werden. Bilddatensätze können durch Importieren neuer Daten (über Import Jobs oder DICOMweb STOW-RS) oder durch Aufrufen erstellt werden. CopyImageSet HealthImaging verschiebt Bilddatensätze, auf die an 30 aufeinanderfolgenden Tagen nicht zugegriffen wurde, automatisch in die Stufe Archive Instant Access. Bilddatensätze verbleiben in der Stufe Archive Instant Access, bis erneut auf sie zugegriffen wird.

Die folgenden Aktionen stellen den Zugriff auf DICOM-Daten dar und verschieben Bilddatensätze automatisch von der Stufe Archive Instant Access zurück in die Stufe für den häufigen Zugriff:

- Aufrufen GetImageSetMetadataoder einer beliebigen DICOMweb WADO-RS-Aktion. GetImageFrame
- Aufrufen von oder. CopyImageSetUpdateImageSetMetadata Bei Kopiervorgängen werden nur die replizierten Bilddatensätze dem Status „Häufiger Zugriff“ zugewiesen. Bei der Option „Mit Ziel kopieren“ wird der Ziel-Bilddatensatz in mehreren Ebenen angeordnet.
- Bilddatensatzdaten über die AWS HealthImaging Managementkonsole anzeigen und herunterladen.

Durch die oben genannten Maßnahmen werden sowohl Bilddatensätze auf die Stufe „Häufiger Zugriff“ heraufgestuft als auch verhindert, dass Bilddatensätze der Stufe „Häufiger Zugriff“ für weitere 30 Tage auf die Stufe Archive Instant Access heruntergestuft werden. Der Zugriff auf Bilddatensätze kann über die AWS Management Console oder über programmatische Schnittstellen wie oder erfolgen. AWS CLI AWS SDKs Andere Aktionen stellen keinen Zugriff dar und verschieben daher Objekte nicht automatisch von der Stufe Archive Instant Access zurück in die Stufe für häufigen Zugriff. Im Folgenden finden Sie ein Beispiel, keine endgültige Liste, solcher Aktionen:

- Aktionen für Datenspeicher (CreateDatastoreund GetDatastore)
- Aktionen auflisten (DICOMImportJobs auflisten ListImageSetVersions, und ListTagsForResource)
- Suchaktionen (SearchImageSetsund DICOMweb QIDO-RS-Abfragen)
- Aufrufen GetImageSetvon, oder TagResourceUntagResource
- Operationen löschen

Für importierte Daten HealthImaging wird eine Mindestspeicherdauer von 30 Tagen berechnet. Sie können Ihre Daten jederzeit löschen, aber für jedes Bild, das vor 30 Tagen nach dem Import gelöscht wird, wird für den Rest der Mindestdauer berechnet.

Schätzung des strukturierten Datenspeichers

HealthImaging speichert einige DICOM-Header-Informationen im indizierten Speicher. Dieser strukturierte Speicherverbrauch wird Ihnen unabhängig von der Speicherebene des Bildsatzes in Rechnung gestellt, und diese Gebühren sind additiv. Sie können den Verbrauch von strukturiertem Speicher schätzen, indem Sie die Anzahl der DICOM-Ressourcen in Ihren Datenspeichern mit den indizierten Datensatzgrößen pro Ressource multiplizieren. Die folgenden Werte sind Näherungswerte.

DICOM-Ressource	Indizierte Größen (Byte)
Studie	1024
Serie	830
Instance	680

HealthImaging AWS-Veröffentlichungen

Die folgende Tabelle zeigt, wann Funktionen und Updates für den HealthImaging AWS-Service und die Dokumentation veröffentlicht wurden. Weitere Informationen zu einer Version finden Sie im verlinkten Thema.

Änderung	Beschreibung	Datum
`includefield` Abfrageparameter für QIDO-RS Such-APIs	HealthImaging unterstützt jetzt den `includefield` Abfrageparameter für alle drei QIDO-RS Such-APIs (`SearchDICOMStudies` `SearchDICOMSeries` , und`SearchDICOMInstances`). Mit dieser Funktion können Sie zusätzliche DICOM-Attribute anfordern , die über den Standardantwortsatz hinausgehen, darunter Standard-Tags, SQ-Attribute (Nested Sequence) mit gepunkteter Pfadnotation und private Datenelemente. Sie können auch angeben`includefield=all` , dass alle verfügbaren Attribute auf einer bestimmten Ressource nebene abgerufen werden sollen. Weitere Informationen finden Sie unter Suchen nach DICOM-Daten.	22. Mai 2026

Import von DICOM-JSON-Metadaten für `StartDICOMImportJob`	HealthImaging fügt den `importConfiguration` Parameter zur `StartDICOMImportJob` Aktion hinzu. Sie können jetzt vorhandene DICOM-Dateien beim Import um JSON-Metadaten erweitern, indem Sie eine angeben`DicomMetadataMapping`, die DICOM-Dateien den entsprechenden JSON-Metadatendateien zuordnet. Weitere Informationen finden Sie unter StartDICOMImportJob.	20. Mai 2026
Updates auf Atomstudienniveau für `UpdateImageSetMetadata`	HealthImaging fügt der Aktion die `--include-study-image-sets` Flagge hinzu. `UpdateImageSetMetadata` Wenn diese Option aktiviert ist, werden Patienten- und Study-level Attributänderungen, die am angegebenen Bilddatensatz vorgenommen wurden, automatisch auf alle anderen primären Bilddatensätze angewendet, die dieselbe Studieninstanz-UID im Datenspeicher verwenden. Weitere Informationen finden Sie unter Bilddatensatz-Metadaten aktualisieren.	31. März 2026

Study-level feinkörnige Zugangskontrolle	HealthImaging unterstützt jetzt die Erteilung von Berechtigungen für DICOMWeb-APIs mithilfe von DICOM Study Instance UIDs und Series Instance UIDs direkt in IAM-Richtlinien. Weitere Informationen finden Sie unter Erteilen von Berechtigungen auf der Grundlage von Study Instance UID und Series Instance UID.	19. März 2026
CloudWatch Amazon-Metriken für HealthImaging	HealthImaging veröffentlicht zusätzliche Metriken CloudWatch im `AWS/HealthImaging` Namespace, einschließlich Kennzahlen zur Ressourcennutzung sowohl auf Konto- als auch auf Datenspeicherebene. Weitere Informationen finden Sie unter Amazon verwenden CloudWatch mit HealthImaging.	12. Februar 2026

Service-linked Rolle für HealthImaging	HealthImaging unterstützt jetzt dienstbezogene Rollen, die es dem Service ermöglichen, Datenspeicher-Metriken in Ihrem Namen zu CloudWatch veröffentlichen. Die `AWSServiceRoleForHealthImaging` Rolle kann mit der IAM-Konsole oder der AWS-CLI erstellt werden. Weitere Informationen finden Sie unter Verwenden von serviceverknüpften Rollen für HealthImaging.	9. Februar 2026
AWSHealthImagingServiceRolePolicy Verwaltete Richtlinie	HealthImaging hat eine neue verwaltete Richtlinie `AWSHealthImagingServiceRolePolicy` für die dienstbezogene Rolle hinzugefügt, die Berechtigungen zur Verwaltung von Dienstvorgängen und zur Veröffentlichung von Servicemetriken für gewährt. CloudWatch Weitere Informationen finden Sie unter AWSHealthImagingServiceRolePolicy.	9. Februar 2026

JPEG XL-Unterstützung für HealthImaging AWS-Datenspeicher	HealthImaging unterstützt das Importieren und Speichern von verlustbehafteten DICOM-Dateien in der JPEG XL-Übertragungssyntax (1.2.840.10008.1.2.4.112). Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen.	2. Februar 2026

Verbesserter DICOM-Import mit Unterstützung für Warnungen bei fehlerhaften Daten	HealthImaging importiert jetzt zuvor abgelehnte DICOM-Daten, indem fehlerhafte Dateien akzeptiert werden und gleichzeitig bei Ereignissen detaillierte Warnungen ausgegeben werden. EventBridge Importaufträge generieren jetzt einen WARNING-Ordner, der `warning.ndjson` Dateien mit spezifischen Codes für Warngründe für fehlerhafte Daten enthält. Das Update bietet Unterstützung für durchsuchbare WarningReason (0008.1196) DICOM-Elemente und führt `transfer-syntax=*` Parameter für das Abrufen von Instanzen in ihrem gespeicherten Format ein, wenn eine Transcodierung nicht möglich ist. Weitere Informationen finden Sie unter Grundlegendes zu Importaufträgen und Warncodes. HealthImaging	8. Dezember 2025

JPEG 2000 Lossless-Unterstützung für HealthImaging AWS-Datenspeicher	AWS unterstützt HealthImaging jetzt die native JPEG 2000-Lossless-Kodierung für medizinische Bilder, sodass Sie persistente Datenspeicher erstellen und verlustfreie Bildframes im JPEG 2000-Format ohne Transcodierung abrufen können. Dies ermöglicht den Abruf mit geringerer Latenz für Anwendungen, die das Format JPEG 2000 Lossless benötigen, wenn Sie dies bei der Erstellung eines Datenspeichers angeben—`lossless-storage-format JPEG_2000_LOSSLESS` .	25. November 2025

QIDO-RS Design für Verbesserungen bei der Suche	HealthImaging Unterstützt jetzt die Platzhaltersuche nach wichtigen DICOM-Attributen (Patientenname, Name des überweisenden Arztes, Patienten-ID, Studienbeschreibung, Modalität und Zugangsnummer) sowie Funktionen zur unscharfen Suche nach Patient/Referring Arztnamen, um unvollständige oder falsch geschriebene Begriffe zu berücksichtigen. Das Update erweitert auch die QIDO-RS API-Abfragefunktionen um die Suche nach dem Geburtsdatum des Patienten und der Studienbeschreibung, wodurch die Suche nach relevanten Studien und Serien verbessert wird. Weitere Informationen finden Sie unter DICOM-Daten durchsuchen in HealthImaging.	15. September 2025

OpenID Connect (OIDC) - Autorisierung für DICOMWeb-APIs	HealthImaging unterstützt jetzt die OpenID Connect (OIDC) -Token-Autorisierung für alle DICOMWeb-APIs. Dies erhöht die standardbasierte OAuth 2.0-Interoperabilität mit gängigen Viewern und Toolkits (z. B. OHIF, SLIM, MONAI), indem JWTs im Header akzeptiert werden. IdP-issued `Authorization` Weitere Informationen finden Sie unter OIDC-Authentifizierung für DICOMWeb-APIs.	3. September 2025
Support für DICOMWeb-Datenimporte und DICOMweb Bulkdata	HealthImaging HealthImaging unterstützt das Speichern von DICOM P10-Dateien über das DICOMWeb-Protokoll. STOW-RS Weitere Informationen finden Sie unter Daten importieren. HealthImaging Unterstützt außerdem DICOM Bulkdata, um konsistente Metadatenabrufe mit geringer Latenz zu gewährleisten. Weitere Informationen finden Sie unter DICOM-Bulkdata abrufen.	30. Juni 2025

Automatische Datenorganisation, DICOMWeb-Suche und DICOMWeb-Erweiterungen QIDO-RS WADO-RS	HealthImaging ermöglicht die automatische Organisation von DICOM P10-Daten beim Import entsprechend der Hierarchie auf Patienten-, Studien- und Serienebene des DICOM-Standards. Weitere Informationen finden Sie unter Grundlegendes zu Importaufträgen. HealthImaging unterstützt Rich Search gemäß dem DICOMWeb-Standard QIDO-RS . Weitere Informationen finden Sie unter DICOM-Daten durchsuchen in HealthImaging. HealthImaging unterstützt das Abrufen der Metadaten für alle DICOM-Instanzen einer Serie über eine einzige API-Aktion, wie unter Metadaten der DICOM-Serie abrufen von beschrieben. HealthImaging	22. Mai 2025
Bei der Erstellung von Bilddatensätzen werden weniger DICOM-Elemente verwendet	HealthImaging verringert die Anzahl der Elemente, die verwendet werden, wenn eingehende DICOM P10-Objekte zu Bilddatensätzen gruppiert werden. Weitere Informationen finden Sie unter Was ist ein Bildsatz? .	27. Januar 2025

Verlustbehaftete Unterstützung für StartDICOMImportJob	HealthImaging unterstützt das Importieren und Speichern von verlustbehafteten DICOM-Dateien (1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50) und binärer Segmentierungsdateien in ihrem Originalformat. Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen.	1. November 2024
Verlustbehaftete Unterstützung für DICOMWeb-Abruf-APIs	HealthImaging unterstützt das Abrufen von Bildern und Instanzen, die in 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 und 1.2.840.10008.1.2.1 (nur binäre Segmentierung) gespeichert sind, in ihrem Originalformat oder in Explicit VR Little Endian (1.2.840.10008.1.2.1). Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen und Abrufen von DICOM-Daten.	1. November 2024
Schnellere Importe für die digitale Pathologie	HealthImaging unterstützt bis zu 6-mal schnellere Importjobs für DICOM Digital Pathology (WSI).	1. November 2024

Unterstützung für binäre Segmentierung	HealthImaging unterstützt sowohl die Aufnahme als auch den Abruf von binären DICOM-Segmentierungsdateien. Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen.	1. November 2024
Kehren Sie zu einer früheren Versions-ID des Imagesets zurück	HealthImaging stellt den `revertToVersionId` Parameter bereit, mit dem die Versions-ID eines früheren Bildsatzes wiederhergestellt werden kann. Weitere Informationen finden Sie `revertToVersionId` in der HealthImaging AWS-API-Referenz.	24. Juli 2024

Erzwingen Sie die Funktionalität für die Änderung des Bilddatensatzes	HealthImaging stellt den `Overrides` Datentyp mit dem optionalen `forced` Anforderungsparameter bereit. Die Einstellung dieses Parameters erzwingt die `CopyImageSet` Aktionen `UpdateImageSetMetadata` `data` und, auch wenn Metadaten auf Patienten-, Studien- oder Serienebene nicht übereinstimmen. Weitere Informationen finden Sie unter Overrides in der HealthImaging AWS-API-Referenz.
	
	• `UpdateImageSetMetadata` `data` Force-Funktionalität — HealthImaging führt den optionalen `force` Anforderungsparameter für die Aktualisierung der folgenden Attribute ein:
	  • `Tag.StudyInstanceUID` , `Tag.SeriesInstanceUID` , `Tag.SOPInstanceUID` , und `Tag.StudyID`
	  • Private DICOM-Datenelemente auf Instanzebene hinzufügen, entfernen oder aktualisieren
	
	Weitere Informationen finden Sie UpdateImageSetMetadata in der
		24. Juli 2024

HealthImaging AWS-API-Referenz.

- `CopyImageSet` Force-Funktionalität — HealthImaging führt den optionalen `force` Anforderungsparameter zum Kopieren von Bilddatensätzen ein. Die Einstellung dieses Parameters erzwingt die `CopyImageSet` Aktion auch dann, wenn Metadaten auf Patienten-, Studien- oder Serienebene im `sourceImageSet` und `destinationImageSet` nicht übereinstimmen. In diesen Fällen bleiben die inkonsistenten Metadaten in der `destinationImageSet` unverändert. Weitere Informationen finden Sie CopyImageSetin der HealthImaging AWS-API-Referenz.

Kopieren Sie Teilmengen von SOP-Instanzen	HealthImaging erweitert die `CopyImageSet` Aktion, sodass Sie eine oder mehrere SOP-Instanzen von a auswählen können`sourceImageSet` , um sie in eine `destinationImageSet` zu kopieren. Weitere Informationen finden Sie unter Kopieren eines Bilddatensatzes.	24. Juli 2024
`GetDICOMInstanceMetadata` für die Rückgabe von DICOM-Instanz-Metadaten	HealthImaging stellt die `GetDICOMInstanceMetadata` API zur Rückgabe von DICOM Part 10-Metadaten (`.json`Datei) bereit. Weitere Informationen finden Sie unter Instanz-Metadaten abrufen.	11. Juli 2024
`GetDICOMInstanceFrames` für die Rückgabe von DICOM-Instanz-Frames (Pixeldaten)	HealthImaging stellt die `GetDICOMInstanceFrames` API zur Rückgabe von DICOM Part 10-Frames (`multipart` Anfrage) bereit. Weitere Informationen finden Sie unter Instanz-Frames abrufen.	11. Juli 2024

Verbesserte Unterstützung für nicht standardmäßige DICOM-Datenimporte	HealthImaging bietet Unterstützung für Datenimporte, die Abweichungen vom DICOM-Standard beinhalten. Weitere Informationen finden Sie unter Einschränkungen für DICOM-Elemente.
	• Die folgenden DICOM-Datenelemente können eine maximale Länge von bis zu 256 Zeichen haben:
	  • `Patient's Name (0010,0010)`
	  • `Patient ID (0010,0020)`
	  • `Accession Number (0008,0050)`
	• Die folgenden Syntaxvariationen sind für `Study Instance UID`, `Series Instance UID`, `Treatment Session UID` `Manufacturer's Device Class UID` `Device UID`, und `Acquisition UID` zulässig:
	  • Das erste Element jeder UID kann Null sein
	  • UIDs können mit einer oder mehreren führenden Nullen beginnen
	  • UIDs können bis zu 256 Zeichen lang sein
		28. Juni 2024

Ereignis-Benachrichtigungen	HealthImaging lässt sich in Amazon integrieren EventBridge , um ereignisgesteuerte Anwendungen zu unterstützen. Weitere Informationen finden Sie unter Verwenden von. EventBridge	5. Juni 2024
`GetDICOMInstance` für die Rückgabe von DICOM-Instanzdaten	HealthImaging stellt den `GetDICOMInstance` Dienst zur Rückgabe von DICOM Part 10-Instanzdaten (`.dcm`Datei) bereit. Weitere Informationen finden Sie unter Eine Instanz abrufen.	15. Mai 2024
Cross-account importieren	HealthImaging bietet Unterstützung für Datenimporte aus Amazon S3 S3-Buckets, die sich in anderen unterstützten Regionen befinden. Weitere Informationen finden Sie unter Cross-accountImport.	15. Mai 2024

Verbesserungen bei der Suche nach Bilddatensätzen	HealthImaging `SearchImageSets` action unterstützt die folgenden Suchverbesserungen. Weitere Informationen finden Sie unter Suchen nach Bilddatensätzen. • Zusätzliche Unterstützung für die Suche nach `UpdatedAt` und `SeriesInstanceUID` • Suchen Sie zwischen Startzeit und Endzeit • Sortiere die Suchergebnisse nach `Ascending` oder `Descending` • Parameter der DICOM-Serie werden in Antworten zurückgegeben	3. April 2024
Die maximale Dateigröße für Importe wurde erhöht	HealthImaging unterstützt eine maximale Dateigröße von 4 GB für jede DICOM P10-Datei in einem Importauftrag. Weitere Informationen finden Sie unter Servicekontingente.	6. März 2024

Übertragungssyntaxen für JPEG Lossless und HTJ2K	HealthImaging unterstützt die folgenden Übertragungssyntaxen für Jobimporte. Weitere Informationen finden Sie unter Unterstützte Übertragungssyntaxen. • 1.2.840.10008.1.2.4.57 — JPEG Lossless (Prozess 14) Non-Hierarchical • 1.2.840.10008.1.2.4.201 — JPEG 2000-Bildkomprimierung (nur verlustfrei) High-Throughput • 1.2.840.10008.1.2.4.202 — JPEG 2000 mit RPCL-Optionen Bildkomprimierung (nur verlustfrei) High-Throughput • 1.2.840.10008.1.2.4.203 — High-Throughput JPEG 2000-Bildkomprimierung	16. Februar 2024
Getestete Codebeispiele	HealthImaging Die Dokumentation enthält getestete Codebeispiele AWS CLI und AWS SDKs für Python JavaScript, Java und C++. Weitere Informationen finden Sie unter Codebeispiele.	19. Dezember 2023
Die maximale Dateianzahl für Importe wurde erhöht	HealthImaging unterstützt bis zu 5.000 Dateien für einen einzigen Importauftrag. Weitere Informationen finden Sie unter Servicekontingente.	19. Dezember 2023

Verschachtelte Ordner für Importe	HealthImaging unterstützt bis zu 10.000 verschachtelte Ordner für einen einzelnen Importauftrag. Weitere Informationen finden Sie unter Dienstkontingente.	1. Dezember 2023
Schnellere Importe	HealthImaging bietet 20-mal schnellere Importe in allen unterstützten Regionen. Weitere Informationen finden Sie unter Dienstendpunkte.	1. Dezember 2023
CloudFormation Unterstützung	HealthImaging unterstützt Infrastructure as Code (IaC) für die Bereitstellung von Datenspeichern. Weitere Informationen finden Sie unter HealthImaging Ressourcen erstellen mit. CloudFormation	21. September 2023
Allgemeine Verfügbarkeit	AWS HealthImaging ist für alle Kunden in den Regionen USA Ost (Nord-Virginia), USA West (Oregon), Europa (Irland) und Asien-Pazifik (Sydney) verfügbar. Weitere Informationen finden Sie unter Service-Endpunkte.	26. Juli 2023

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.