



Entwicklerhandbuch

AWS HealthImaging



AWS HealthImaging: Entwicklerhandbuch

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Handelsmarken und 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	3
Zugriff	4
HIPAA	5
Preisgestaltung	5
Erste Schritte	6
Konzepte	6
Datastore	6
Bildset	7
Metadaten	7
Bildrahmen	7
Einrichtung	8
Melde dich an für eine AWS-Konto	8
Erstellen eines Benutzers mit Administratorzugriff	9
S3-Buckets erstellen	10
Einen Datenspeicher erstellen	11
Erstellen eines IAM-Benutzers	11
Erstellen Sie eine IAM-Rolle	12
Installiere das AWS CLI	14
Tutorial	15
Verwaltung von Datenspeichern	17
Einen Datenspeicher erstellen	17
Eigenschaften des Datenspeichers abrufen	24
Datenspeicher auflisten	30
Löschen eines Datenspeichers	37
Grundlegendes zu Speicherstufen	43
Bilddaten importieren	46
Importaufträge verstehen	46
Einen Importjob starten	49
Eigenschaften von Importaufträgen abrufen	57
Importaufträge auflisten	63
Zugreifen auf Bilddatensätze	69

Bilddatensätze verstehen	69
Suche nach Bilddatensätzen	75
Eigenschaften von Bilddatensätzen abrufen	100
Metadaten von Bilddatensätzen abrufen	105
Pixeln des Bildsatzes abrufen	115
Ändern von Bilddatensätzen	122
Versionen von Bildsätzen auflisten	122
Metadaten von Bilddatensätzen aktualisieren	128
Kopieren eines Bilddatensatzes	146
Löschen eines Bilddatensatzes	161
Markieren von Ressourcen	167
Eine Ressource taggen	167
Tags für eine Ressource auflisten	172
Eine Ressource entkennzeichnen	176
Codebeispiele	182
Grundlagen	188
Hallo HealthImaging	189
Aktionen	195
Szenarien	331
Erste Schritte mit Bildsätzen und Bildrahmen	332
Kennzeichnen eines Datenspeichers	386
Einen Bilddatensatz taggen	396
DICOMweb	407
Abrufen von Daten	407
Eine Instanz abrufen	409
Instanz-Metadaten abrufen	410
Instanzframes abrufen	412
Überwachen	414
Benutzen CloudTrail	415
Erstellen eines Trails	415
Logeinträge verstehen	417
Verwenden CloudWatch	418
HealthImaging Metriken anzeigen	419
Erstellen eines Alarms	420
Verwenden EventBridge	420
HealthImaging Ereignisse wurden gesendet an EventBridge	420

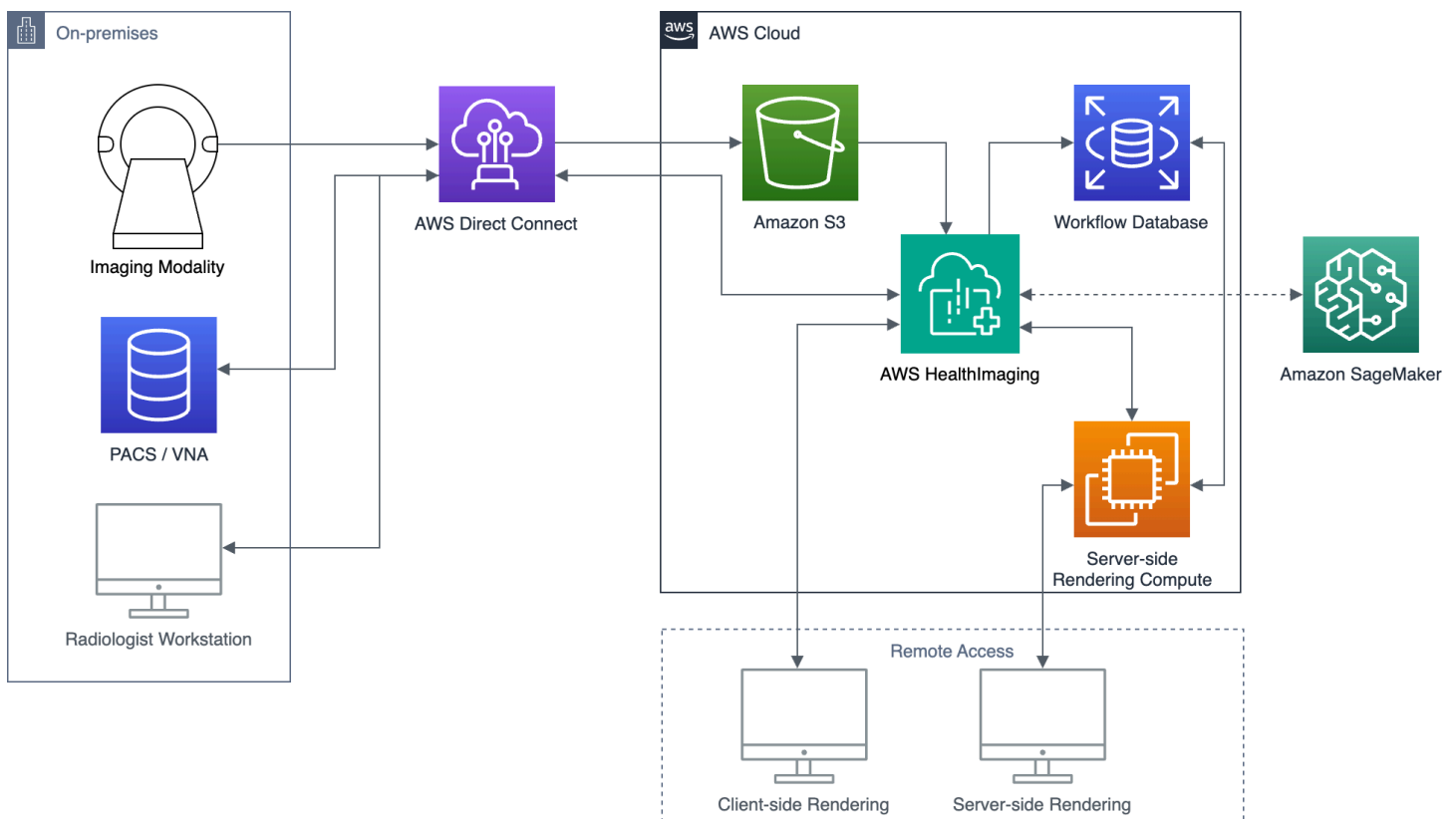
HealthImaging Struktur und Beispiele für Ereignisse	422
Sicherheit	438
Datenschutz	439
Datenverschlüsselung	440
Datenschutz im Netzwerkverkehr	450
Identitäts- und Zugriffsverwaltung	450
Zielgruppe	451
Authentifizierung mit Identitäten	451
Verwalten des Zugriffs mit Richtlinien	456
Wie AWS HealthImaging funktioniert mit IAM	458
Beispiele für identitätsbasierte Richtlinien	466
Von AWS verwaltete Richtlinien	469
Fehlerbehebung	472
Compliance-Validierung	474
Sicherheit der Infrastruktur	475
Infrastruktur als Code	475
HealthImaging und AWS CloudFormation Vorlagen	476
Weitere Informationen zu AWS CloudFormation	476
VPC-Endpunkte	476
Überlegungen zu VPC-Endpunkten	477
Erstellung eines VPC-Endpunkts	477
Erstellen einer VPC-Endpunktrichtlinie	478
Kontoübergreifender Import	479
Ausfallsicherheit	481
Referenz	482
DICOMUnterstützung	482
Unterstützte Klassen SOP	483
Normalisierung der Metadaten	483
Unterstützte Übertragungssyntaxen	488
DICOMElementeinschränkungen	489
DICOMEinschränkungen bei Metadaten	491
Überprüfung der Pixeldaten	491
HTJ2K-Decodierungsbibliotheken	493
HTJ2K-Decodierungsbibliotheken	494
Bildbetrachter	494
Endpunkte und Kontingente	494

Service-Endpunkte	494
Servicekontingente	497
Drosselungsgrenzen	500
Beispielprojekte	501
Mit AWS SDKs arbeiten	503
Versionen	505
.....	dxiv

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 unbekanntem Hexadezimalzahlen für Gruppen oder Elemente möglich. DICOM-Elemente auf Patienten-, Studien- und Serienebene sind [normalisiert, sodass sich Anwendungsentwickler nicht mehr](#)

[mit Inkonsistenzen](#) zwischen SOP-Instanzen auseinandersetzen müssen. Darüber hinaus sind Metadaten-Attributwerte in systemeigenen Laufzeittypen direkt zugänglich.

SIMD-beschleunigte Bilddekodierung

AWS HealthImaging gibt Bildframes (Pixeldaten) zurück, die mit High Throughput JPEG 2000 (HTJ2K), einem fortschrittlichen Bildkomprimierungscodec, codiert sind. 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 JPEG2000 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.

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 [DICOMUnterstützung](#)

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 zu 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 AWS CloudFormation , um IaC-Vorlagen (Infrastructure as Code) zu implementieren, in HealthImaging den 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 den AWS Command Line Interface und den AWS Management Console AWS SDKs auf AWS zugreifen. Dieses Handbuch enthält Verfahrensanweisungen für die AWS SDKs AWS Management Console AWS CLI und Codebeispiele.

AWS Management Console

Das AWS Management Console 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 bieten Bibliotheken, Codebeispiele und andere Ressourcen für Softwareentwickler. 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 für 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 Anfragen HTTP](#).

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 unter [Konformitätsvalidierung für AWS HealthImaging](#).

Preisgestaltung

HealthImaging hilft Ihnen, das Lebenszyklusmanagement klinischer Daten mit intelligentem Tiering zu automatisieren. Weitere Informationen finden Sie unter [Grundlegendes zu Speicherstufen](#).

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 CLI](#) oder die [AWS SDKs](#) verwenden zu können, 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](#)
- [einrichten AWS HealthImaging](#)
- [AWS HealthImaging Tutorial](#)

HealthImaging AWS-Konzepte

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

Konzepte

- [Datastore](#)
- [Bildset](#)
- [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 [Grundlegendes zu Speicherstufen](#).

Bildset

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

Weitere Informationen erhalten Sie unter [Bilddaten 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 KI/ML-Anwendungsfälle, die keine Pixeldaten benötigen. 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 der Metadaten](#).

Bildrahmen

Ein Bildrahmen sind die Pixeldaten, die in einem [Bilddatensatz](#) vorhanden sind, um ein medizinisches 2D-Bild zu erstellen. Während des Imports HealthImaging codiert AWS alle Bildrahmen in High-Throughput JPEG 2000 (HTJ2K). Daher müssen Bildrahmen vor der Anzeige dekodiert werden.

Weitere Informationen finden Sie unter [Pixeldaten des Bildsatzes abrufen](#) und [HTJ2K-Decodierbibliotheken](#).

einrichten AWS HealthImaging

Sie müssen Ihre einrichten AWS Umgebung vor der Verwendung AWS HealthImaging. Die folgenden Themen sind Voraussetzungen für das [Tutorial](#) im nächsten Abschnitt.

Themen

- [Melde dich an für eine AWS-Konto](#)
- [Erstellen eines Benutzers mit Administratorzugriff](#)
- [S3-Buckets erstellen](#)
- [Einen Datenspeicher erstellen](#)
- [Erstellen Sie einen IAM Benutzer mit HealthImaging voller Zugriffsberechtigung](#)
- [Erstellen Sie eine IAM Rolle für den Import](#)
- [Installieren Sie den AWS CLI \(Optional\)](#)

Melde dich an für eine AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um einen zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie <https://portal.aws.amazon.com/billing/die-Anmeldung>.
2. Folgen Sie den Online-Anweisungen.

Bei der Anmeldung müssen Sie auch einen Telefonanruf entgegennehmen und einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, ein Root-Benutzer des AWS-Kontos wird erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen im Konto. Als bewährte Sicherheitsmethode weisen Sie einem Administratorbenutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um [Aufgaben auszuführen, die Root-Benutzerzugriff erfordern](#).

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu <https://aws.amazon.com/> gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie sich Ihre Root-Benutzer des AWS-Kontos, aktivieren AWS IAM Identity Center, und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Loggen Sie sich ein bei [AWS Management Console](#) als Kontoinhaber wählen Sie Root-Benutzer und geben Sie Ihren AWS-Konto E-Mail-Adresse. Geben Sie auf der nächsten Seite Ihr Passwort ein.

Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter [Als Root-Benutzer anmelden in der AWS-Anmeldung Benutzerleitfaden](#).

2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für Ihren Root-Benutzer.

Anweisungen finden Sie unter [Aktivieren Sie ein virtuelles MFA Gerät für Ihr AWS-Konto Root-Benutzer \(Konsole\)](#) im IAMBenutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie IAM Identity Center.

Anweisungen finden Sie unter [Aktivieren AWS IAM Identity Center](#) in der AWS IAM Identity Center Benutzerleitfaden.

2. Gewähren Sie einem Benutzer in IAM Identity Center Administratorzugriff.

Für ein Tutorial zur Verwendung des IAM-Identity-Center-Verzeichnis als Identitätsquelle finden [Sie unter Benutzerzugriff mit der Standardeinstellung konfigurieren IAM-Identity-Center-Verzeichnis](#) in der AWS IAM Identity Center Benutzerleitfaden.

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM Identity Center-Benutzer anzumelden, verwenden Sie die Anmeldung, URL die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM Identity Center-Benutzer erstellt haben.

Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie unter [Anmelden bei AWS Zugriffsportal](#) im AWS-Anmeldung Benutzerleitfaden.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie in IAM Identity Center einen Berechtigungssatz, der der bewährten Methode zur Anwendung von Berechtigungen mit den geringsten Rechten folgt.

Anweisungen finden Sie unter [Einen Berechtigungssatz erstellen in](#) der AWS IAM Identity Center Benutzerleitfaden.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

Eine Anleitung finden [Sie unter Gruppen hinzufügen](#) im AWS IAM Identity Center Benutzerleitfaden.

S3-Buckets erstellen

Um DICOM P10-Daten in zwei Amazon S3-Buckets zu importieren 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 unter [Importaufträge verstehen](#).

Note

Aufgrund AWS Identity and Access Management (IAM) Richtlinie, Ihre Amazon S3 S3-Bucket-Namen müssen 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 [IAMRolle für den Import](#).

- Eingabe-Bucket: `arn:aws:s3:::medical-imaging-dicom-input`
- Ausgabe-Bucket: `arn:aws:s3:::medical-imaging-output`

Weitere Informationen finden Sie unter [Erstellen eines Buckets](#) im Amazon S3 S3-Benutzerhandbuch.

Einen Datenspeicher erstellen

Wenn Sie Ihre medizinischen Bilddaten importieren, enthält der AWS HealthImaging [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 `verwendendatastoreID`, wenn Sie das [trust relationship](#) für den Import später in diesem Abschnitt abschließen.

Informationen zum Erstellen eines Datenspeichers finden Sie unter [Einen 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 entspricht [Grant Least-Privilege-Zugriff](#) in AWS Well-Architected Framework.

Für die Zwecke des [Tutorials](#) 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 IAMBenutzerhandbuch. Erwägen Sie zur Verdeutlichung die Benennung des Benutzers `hiadmin` (oder etwas Ähnliches).
2. Weisen Sie dem IAM Benutzer die `AWSHealthImagingFullAccess` verwaltete Richtlinie zu. Weitere Informationen finden Sie unter [AWSverwaltete Richtlinie: AWSHealthImagingFullAccess](#).

Note

IAMBerechtigungen können eingeschränkt werden. Weitere Informationen finden Sie unter [AWSverwaltete Richtlinien für AWS HealthImaging](#).

Erstellen Sie eine IAM Rolle für den Import

Note

Die folgenden Anweisungen beziehen sich auf 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 IAM Berechtigungen hinzuzufügen [AWSverwaltete Richtlinien für AWS HealthImaging](#), da sie einfacher zu verwenden sind, 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. Außerdem muss ihr eine Vertrauensbeziehung (Richtlinie) zugewiesen werden, die es ermöglicht, die AWS HealthImaging Rolle zu übernehmen.

Um eine IAM Rolle für Importzwecke zu erstellen

1. Erstellen Sie in der [IAMKonsole](#) eine Rolle mit dem Namen `ImportJobDataAccessRole`. Sie verwenden diese Rolle für das [Tutorial](#) im nächsten Abschnitt. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter [IAMRollen erstellen](#).

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 `ImportJobDataAccessRole` die folgende Berechtigungsrichtlinie hinzu.

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

3. Ordnen Sie der `ImportJobDataAccessRole` IAM Rolle die folgende Vertrauensbeziehung (Richtlinie) zu. Die Vertrauensrichtlinie erfordert `datastoreId`, die generiert wurde, als Sie den Abschnitt abgeschlossen haben [Einen Datenspeicher erstellen](#). [In der Anleitung](#) zu diesem Thema wird davon ausgegangen, dass Sie einen AWS HealthImaging Datenspeicher

verwenden, jedoch mit datenspeicherspezifischen Amazon S3 S3-Buckets, IAM Rollen und Vertrauensrichtlinien.

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

Weitere Informationen zum Erstellen und Verwenden von IAM Richtlinien mit AWS HealthImaging finden Sie unter [Identitäts- und Zugriffsverwaltung für AWS HealthImaging](#)

Weitere Informationen zu IAM Rollen im Allgemeinen finden Sie unter [IAMRollen](#) im IAMBenutzerhandbuch. Weitere Informationen zu IAM Richtlinien und Berechtigungen im Allgemeinen finden Sie im IAMBenutzerhandbuch unter [IAMRichtlinien und Berechtigungen](#).

Installieren Sie den AWS CLI (Optional)

Das folgende Verfahren ist erforderlich, wenn Sie AWS Command Line Interface. Wenn du das verwendest AWS Management Console or AWS SDKs, können Sie das folgende Verfahren überspringen.

Um das einzurichten AWS CLI

1. Laden Sie das herunter und konfigurieren Sie es AWS CLI. Anweisungen finden Sie in den folgenden Themen im AWS Command Line Interface Benutzerleitfaden.
 - [Installation oder Aktualisierung der neuesten Version von AWS CLI](#)
 - [Erste Schritte mit dem AWS CLI](#)
2. Im AWS CLI configDatei, fügen Sie ein benanntes Profil für den Administrator hinzu. Sie verwenden dieses Profil beim Ausführen von AWS CLI Befehle. 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 unter [Einstellungen für Konfiguration und Anmeldeinformationsdateien](#) im AWS Command Line Interface Benutzerleitfaden.

```
[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 das Symbol AWS CLI ist korrekt konfiguriert, Sie sehen eine kurze Beschreibung AWS HealthImaging und eine Liste der verfügbaren Befehle.

AWS HealthImaging Tutorial

Ziel

Das Ziel dieses Tutorials besteht darin, DICOM P10-Binärdateien (.dcmDateien) in einen AWS HealthImaging [Datenspeicher](#) zu importieren und sie in [Bilddateien](#) 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 Bilddateien, 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 Management Console AWS CLI, aufzulisten und zu löschen AWS SDKs.

Note

Das letzte Thema in diesem Kapitel befasst sich mit dem [Verständnis von Speicherebenen](#). 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 Tierverschiebung und die HealthImaging Ressourcen zu verstehen, die für Abrechnungszwecke anerkannt werden.

Themen

- [Einen Datenspeicher erstellen](#)
- [Eigenschaften des Datenspeichers abrufen](#)
- [Datenspeicher auflisten](#)
- [Löschen eines Datenspeichers](#)
- [Grundlegendes zu Speicherstufen](#)

Einen Datenspeicher erstellen

Verwenden Sie die `CreateDatastore` Aktion, um einen AWS HealthImaging [Datenspeicher](#) für den Import von DICOM P10-Dateien zu erstellen. Die folgenden Menüs bieten ein Verfahren für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [CreateDatastore](#) in der AWS HealthImaging APIReferenz.

Wichtig

Benennen Sie keine Datenspeicher mit geschützten Gesundheitsinformationen (PHI), personenbezogenen Daten (PII) oder anderen vertraulichen oder sensiblen Informationen.

Um einen Datenspeicher zu erstellen

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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 eine AWS KMS Schlüssel zum Verschlüsseln Ihrer Ressourcen. 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 getopt command in a function.

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

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

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

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

    local error_code=${?}
```

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

echo "$response"

return 0
}
```

- API-Einheiten finden Sie in [CreateDatastore](#) AWS CLI Befehlsreferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen Datenspeicher zu erstellen

Das folgende `create-datastore` Codebeispiel erstellt einen Datenspeicher mit dem Namen `my-datastore`.

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

Ausgabe:

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

Weitere Informationen finden Sie unter [Erstellen eines Datenspeichers](#) im AWS HealthImaging Entwicklerhandbuch.

- APIEinzelheiten finden Sie [CreateDatastore](#) in 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 "";
}
```

- APIEinzelheiten finden Sie in [CreateDatastore](#) AWS SDK for Java 2.x APIReferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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

- API-Einheiten finden Sie [CreateDatastore](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [CreateDatastore](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Eigenschaften des Datenspeichers abrufen

Verwenden Sie die `GetDatastore` Aktion, um die Eigenschaften des AWS HealthImaging [Datenspeichers](#) abzurufen. Die folgenden Menüs bieten ein Verfahren für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [GetDatastore](#) in der AWS HealthImaging APIReferenz.

Um die Eigenschaften des Datenspeichers abzurufen

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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 getopt command in a
    function.
    local error_code
    # bashsupport disable=BP5008
    function usage() {
        echo "function imaging_get_datastore"
        echo "Gets a data store's properties."
        echo " -i datastore_id - The ID of the data store."
        echo ""
    }

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

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

```

```
local response

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

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

echo "$response"

return 0
}
```

- API-Einheiten finden Sie in [GetDatastore](#) AWS CLI Befehlsreferenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um die Eigenschaften eines Datenspeichers abzurufen

Das folgende `get-datastore` Codebeispiel ruft die Eigenschaften eines Datenspeichers ab.

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



```
--datastore-id 12345678901234567890123456789012
```

Ausgabe:

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

Weitere Informationen finden Sie unter [Abrufen der Eigenschaften eines Datenspeichers](#) in AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [GetDatastore](#) in 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;
}
```

- API-Einheiten finden Sie in [GetDatastore](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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"];  
};
```

- API-Einheiten finden Sie [GetDatastore](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 Objekt. MedicalImagingWrapper

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

- API-Einheiten finden Sie unter [GetDatastore](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

Datenspeicher auflisten

Verwenden Sie die `ListDatastores` Aktion, um verfügbare [Datenspeicher](#) in aufzulisten AWS HealthImaging. Die folgenden Menüs bieten eine Vorgehensweise für AWS Management Console und Codebeispiele für AWS CLI und AWS SDKs. Weitere Informationen finden Sie [ListDatastores](#) in der AWS HealthImaging APIReferenz.

Um Datenspeicher aufzulisten

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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 getopt command in a function.  
    local error_code  
    # bashsupport disable=BP5008  
    function usage() {  
        echo "function imaging_list_datastores"  
        echo "Lists the AWS HealthImaging data stores in the account."  
        echo ""  
    }  
  
    # Retrieve the calling parameters.  
    while getopt "h" option; do  
        case "${option}" in  
            h)  
                usage  
                return 0  
                ;;  
        esac  
    done  
}
```

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

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

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

echo "$response"

return 0
}
```

- API-Einheiten finden Sie in [ListDatastores](#) AWS CLI Befehlsreferenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um Datenspeicher aufzulisten

Das folgende `list-datastores` Codebeispiel listet die verfügbaren Datenspeicher auf.

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 unter [Auflisten von Datenspeichern](#) im AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [ListDatastores](#) in 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;
}
```

- API-Einheiten finden Sie in [ListDatastores](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 it is
        // larger than `pageSize`.
        datastoreSummaries.push(...page["datastoreSummaries"]);
        console.log(page);
    }
}
```



```

// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   dataStoreSummaries: [
//     {
//       createdAt: 2023-08-04T18:49:54.429Z,
//       dataStoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       dataStoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       dataStoreName: 'my_datastore',
//       dataStoreStatus: 'ACTIVE',
//       updatedAt: 2023-08-04T18:49:54.429Z
//     }
//     ...
//   ]
// }

return dataStoreSummaries;
};

```

- API-Einheiten finden Sie [ListDataStores](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListDatastores](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

Löschen eines Datenspeichers

Verwenden Sie die `DeleteDatastore` Aktion, um einen AWS HealthImaging [Datenspeicher](#) zu löschen. Die folgenden Menüs bieten ein Verfahren für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [DeleteDatastore](#) in der AWS HealthImaging APIReferenz.

Note

Bevor ein Datenspeicher gelöscht werden kann, müssen Sie zuerst alle darin enthaltenen [Bilddatensätze](#) löschen. Weitere Informationen finden Sie unter [Löschen eines Bilddatensatzes](#).

Um einen Datenspeicher zu löschen

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

AWS Konsole

1. Öffnen Sie die [Seite Datenspeicher](#) der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.
3. Wählen Sie Löschen.

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 getopt command in a function.

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

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

```
export OPTIND=1

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

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

local error_code=${?}

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

return 0
}
```

- API-Einheiten finden Sie in [DeleteDatastore](#) AWS CLI Befehlsreferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen Datenspeicher zu löschen

Das folgende `delete-datastore` Codebeispiel löscht einen Datenspeicher.

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

Ausgabe:

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

Weitere Informationen finden Sie unter [Löschen eines Datenspeichers](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [DeleteDatastore](#) in 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);
    }
}
```

- API-Einheiten finden Sie in [DeleteDatastore](#) AWS SDK for Java 2.x API Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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

  return response;
};
```

- API-Einheiten finden Sie [DeleteDatastore](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [DeleteDatastore](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Grundlegendes zu Speicherstufen

AWS HealthImaging verwendet intelligentes Tiering für das automatische Management des klinischen Lebenszyklus. Dies führt zu einer überzeugenden Leistung und einem überzeugenden Preis sowohl für neue oder aktive Daten als auch für Daten zur Langzeitarchivierung ohne Reibungsverluste. HealthImaging berechnet Speicherplatz pro GB/Monat, wobei die folgenden Stufen verwendet werden.

- Stufe für häufigen Zugriff — Eine Stufe für häufig abgerufene Daten.
- Archive Instant Access Tier — Eine Stufe für archivierte Daten.

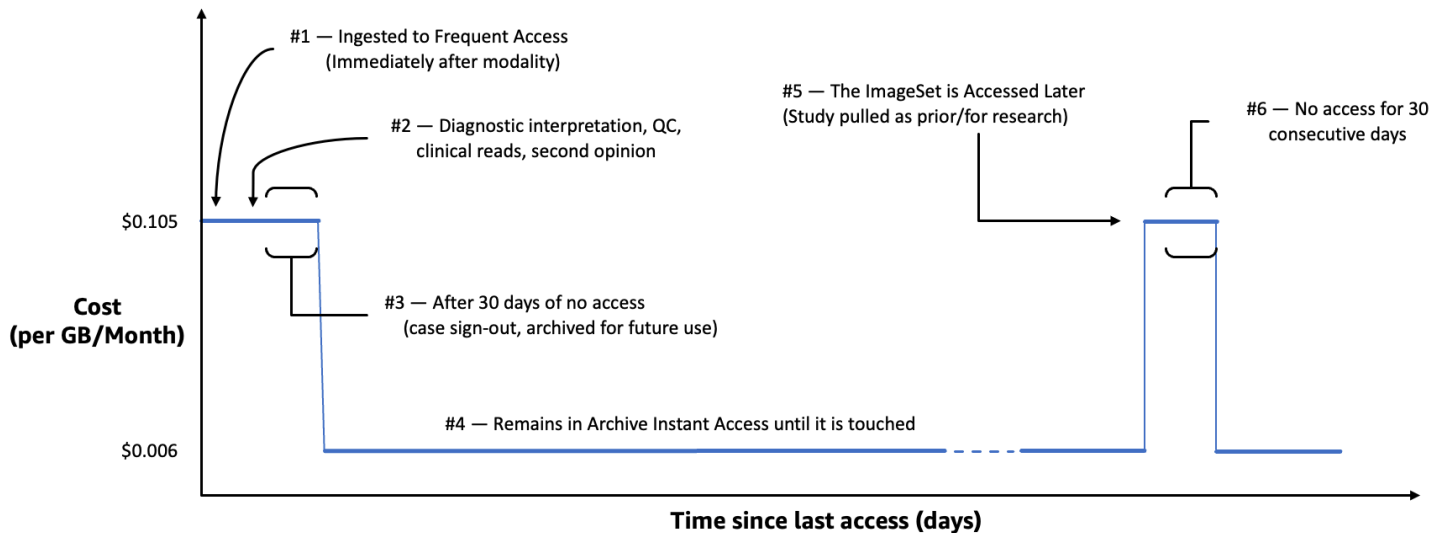
Note

Es gibt keinen Leistungsunterschied zwischen den Stufen „Frequent Access“ und „Archive Instant Access“. Intelligentes Tiering wird auf bestimmte API-Aktionen für [Bilddatensätze](#) angewendet. Intelligentes Tiering erkennt keine API-Aktionen zum Speichern, Importieren und Kennzeichnen von Daten. Der Wechsel zwischen den Stufen erfolgt je nach API-Nutzung automatisch und wird im folgenden Abschnitt erklärt.

Wie funktioniert das Verschieben von Stufen?

- Nach dem Import beginnen die Bilddatensätze in der Stufe für den häufigen Zugriff.
- Nach 30 aufeinanderfolgenden Tagen ohne Bearbeitung werden Bilddatensätze automatisch in den Status Archive Instant Access verschoben.
- Bilddatensätze der Archive-Instant-Access-Stufe werden erst wieder in die Stufe für häufigen Zugriff verschoben, wenn sie berührt werden.

Die folgende Grafik bietet einen Überblick über den HealthImaging intelligenten Tiering-Prozess.




Was wird als Berührung angesehen?

Eine Berührung ist ein bestimmter API-Zugriff über die AWS Management Console AWS CLI, oder AWS SDKs und erfolgt, wenn:

1. Ein neuer Bildsatz wird erstellt (`StartDICOMImportJoboderCopyImageSet`)
2. Ein Bilddatensatz wird aktualisiert (`UpdateImageSetMetadataoderCopyImageSet`)
3. Die zugehörigen Metadaten oder der Bildrahmen (Pixeldaten) eines Bilddatensatzes werden gelesen (`GetImageSetMetaDataoderGetImageFrame`)

Die folgenden HealthImaging API-Aktionen führen zu Berührungen und verschieben Bilddatensätze von der Stufe für den sofortigen Zugriff auf die Stufe für den häufigen Zugriff.

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

 Note

Obwohl [Bildrahmen](#) (Pixeldaten) mit der UpdateImageSetMetadata Aktion nicht gelöscht werden können, werden sie dennoch zu Abrechnungszwecken gezählt.

Die folgenden HealthImaging API-Aktionen führen nicht zu Berührungen. Daher verschieben sie Bilddatensätze nicht von der Stufe für den sofortigen Zugriff auf die Stufe für den häufigen Zugriff.

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

Import von Bilddaten mit AWS HealthImaging

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

Wichtig

HealthImaging Import-Jobs verarbeiten DICOM Instanz-Binärdateien (.dcmDateien) und transformieren sie in Bilddatensätze. Verwenden Sie HealthImaging [Cloud-native Aktionen](#) (APIs), um Datenspeicher und Bildsätze 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 Management Console AWS CLI, und in einen HealthImaging Datenspeicher importieren AWS SDKs.

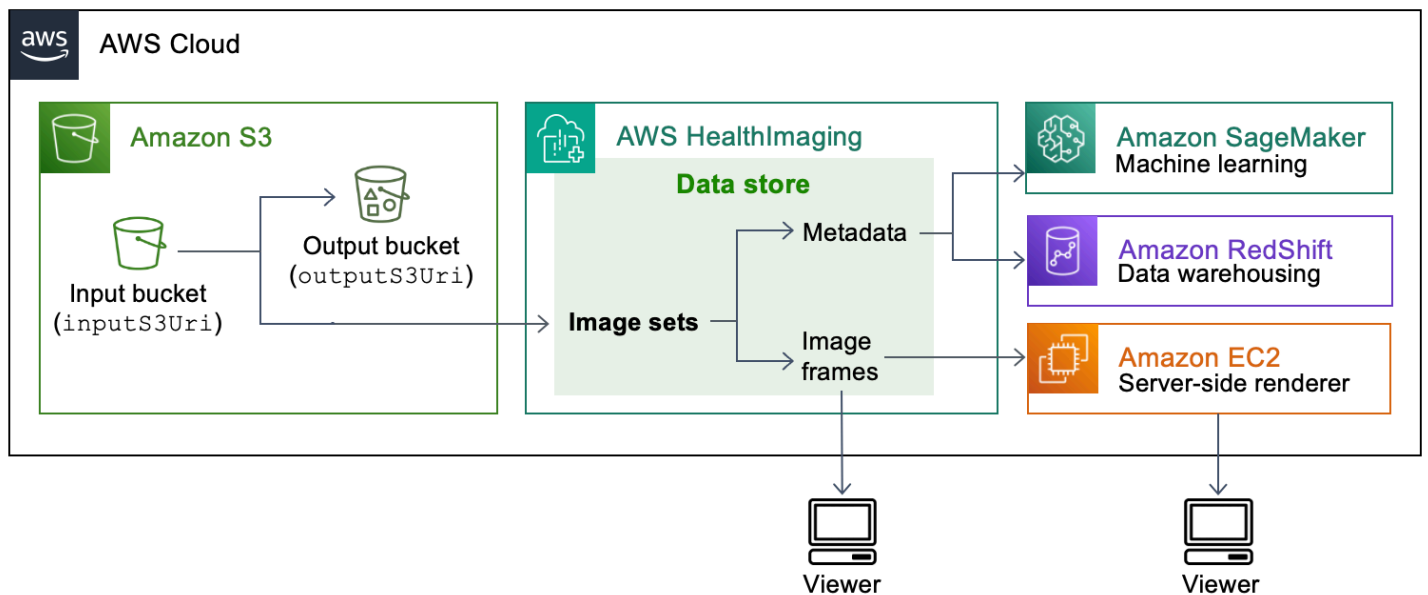
Themen

- [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 Management Console AWS CLI, und verwenden, um Importaufträge zu starten, zu beschreiben und aufzulisten.

Das folgende Diagramm bietet einen Überblick darüber, wie DICOM-Daten in einen Datenspeicher HealthImaging importiert und in Bilddatensätze umgewandelt werden. Die Ergebnisse der Importauftragsverarbeitung werden im Amazon S3 S3-Ausgabe-Bucket (outputS3Uri) gespeichert, und Bildsätze werden im HealthImaging AWS-Datenspeicher gespeichert.



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

- Bestimmte SOP-Klassen und Übertragungssyntaxen werden für Importaufträge unterstützt. Weitere Informationen finden Sie unter [DICOMUnterstützung](#).
- 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 [DICOMElementeinschränkungen](#).
- 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 finden Sie unter [Endpunkte und Kontingente](#) und [Drosselungsgrenzen](#).
- Für jeden Importauftrag werden die Verarbeitungsergebnisse am Standort gespeichert. `outputS3Uri` Die Verarbeitungsergebnisse sind in `job-output-manifest.json` Dateien SUCCESS und FAILURE Ordnern 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/",
    "numberOfScannedFiles": 5,
    "numberOfImportedFiles": 3,
    "numberOfFilesWithCustomerError": 2,
    "numberOfFilesWithServerError": 0,
    "numberOfGeneratedImageSets": 2,
    "imageSetsSummary": [{
      "imageSetId": "12345612345612345678907890789012",
      "numberOfMatchedSOPInstances": 2
    },
    {
      "imageSetId": "12345612345612345678917891789012",
      "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"}}
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105630.1.0.1.dcm","importResponse":{"imageSetId":"12345612345612345678917891789012"}}
```

- 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"}}
```

- 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 AWS HealthImaging Datenspeicher](#) zu starten. Der Importauftrag importiert DICOM P10-Dateien, die sich in dem durch den `inputS3Uri` Parameter angegebenen Amazon S3 S3-Eingabe-Bucket befinden. Die Ergebnisse der Importauftragsverarbeitung werden in dem durch den `outputS3Uri` Parameter angegebenen Amazon S3 S3-Ausgabe-Bucket gespeichert.

Note

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 [Kontoubergreifender Import für AWS HealthImaging](#).

Während des Imports werden Längenbeschränkungen auf bestimmte DICOM Elemente angewendet. Weitere Informationen finden Sie unter [DICOMElementeinschränkungen](#).

Die folgenden Menüs bieten ein Verfahren für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [StartDICOMImportJob](#) in der AWS HealthImaging APIReferenz.

Um einen Importjob zu starten

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

AWS Konsole

1. Öffnen Sie die [Seite Datenspeicher](#) der HealthImaging Konsole.
2. Wählen Sie einen Datenspeicher aus.
3. Wählen Sie DICOMDaten importieren.

Die Seite DICOMDaten 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();
}

```

- API-Einheiten finden Sie unter [StartDICOMImportJob](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen DICOM-Importjob zu starten

Das folgende `start-dicom-import-job` Codebeispiel startet einen DICOM-Importauftrag.

```

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 Leitfaden für Entwickler.

- API-Einheiten finden Sie unter [StartDICOMImport Job](#) in 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 "";
}
```

- API-Einheiten finden Sie unter [StartDICOMImport Job](#) in AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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

```

- API-Einheiten finden Sie unter [Start DICOM Import Job](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [StartDICOMImport Job](#) in AWS SDK für Python (Boto3) API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Eigenschaften von Importaufträgen abrufen

Verwenden Sie die `GetDICOMImportJob` Aktion, um mehr über AWS HealthImaging Importauftragseigenschaften zu erfahren. Beispielsweise können Sie nach dem Start eines Importauftrags ausführen, `GetDICOMImportJob` um den Status des Jobs 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 bieten ein Verfahren für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [GetDICOMImportJob](#) in der AWS HealthImaging APIReferenz.

Um die Eigenschaften eines Importauftrags abzurufen

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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;
}
```

- API Einheiten finden Sie unter [GetDICOMImport Job](#) in AWS SDK for C++ API Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um die Eigenschaften eines DICOM-Importjobs abzurufen

Im folgenden `get-dicom-import-job` Codebeispiel 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 unter [Abrufen der Eigenschaften eines Importauftrags](#) im AWS HealthImaging Leitfadens für Entwickler.

- API-Einheiten finden Sie unter [GetDICOMImport Job](#) in 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;
}
```

- API-Einheiten finden Sie unter [GetDICOMImportJob](#) in AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 = "xxxxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a2637936-78ea-44e7-98b8-7a87d95dface',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   jobProperties: {
  //     dataAccessRoleArn: 'arn:aws:iam:xxxxxxxxxxxx:role/dicom_import',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //     endedAt: 2023-09-19T17:29:21.753Z,
  //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobName: 'job_1',
  //     jobStatus: 'COMPLETED',
  //     outputS3Uri: 's3://health-imaging-dest/
  output_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxx'/',
  //     submittedAt: 2023-09-19T17:27:25.143Z
  //   }
  // }

  return response;
};

```

- APIEinzelheiten finden Sie unter [GetDICOMImport Job](#) in AWS SDK for JavaScript APIReferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

```
medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API-Einheiten finden Sie unter [GetDICOMImportJob](#) in AWS SDK für Python (Boto3) API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Importaufträge auflisten

Verwenden Sie die `ListDICOMImportJobs` Aktion, um Importaufträge aufzulisten, die für einen bestimmten HealthImaging [Datenspeicher](#) erstellt wurden. Die folgenden Menüs bieten ein Verfahren für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [ListDICOMImportJobs](#) in der AWS HealthImaging API-Referenz.

Note

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

Um Importaufträge aufzulisten

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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

Um DICOM-Importaufträge aufzulisten

Das folgende `list-dicom-import-jobs` Codebeispiel listet DICOM-Importaufträge auf.

```
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 Leitfaden für Entwickler.

- Weitere API Informationen finden Sie unter [listDICOMImportL-Jobs](#) in 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<>();
}

```

- API-Einheiten finden Sie unter [ListDICOMImportJobs in AWS SDK for Java 2.x API-Referenz](#).

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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

```

```

    client: medicalImagingClient,
    pageSize: 50,
  };

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

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   jobSummaries: [
  //     {
  //       dataAccessRoleArn: 'arn:aws:iam:xxxxxxxxxxxx:role/
dicom_import',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       endedAt: 2023-09-22T14:49:51.351Z,
  //       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       jobName: 'test-1',
  //       jobStatus: 'COMPLETED',
  //       submittedAt: 2023-09-22T14:48:45.767Z
  //     }
  //   ]
  // }

  return jobSummaries;
};

```

- API-Einheiten finden Sie unter [ListDICOMImport Jobs](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListDICOMImport Jobs in AWS SDK für Python \(Boto3\) API](#) -Referenz.

 Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

Zugreifen auf Bilddatensätze mit AWS HealthImaging

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

Wichtig

HealthImaging verarbeitet beim Import DICOM Instanz-Binärdateien (.dcmDateien) und wandelt sie in Bilddatensätze um. Verwenden Sie HealthImaging [Cloud-native Aktionen](#) (APIs), um Datenspeicher und Bildsätze 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 Management Console, 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

Bildsätze sind ein AWS Konzept, das als Grundlage für AWS dient HealthImaging. Bildsätze werden erstellt, wenn Sie Ihre DICOM-Daten importieren. Daher ist es erforderlich HealthImaging, dass Sie bei der Arbeit mit dem Service ein gutes Verständnis dieser Datensätze haben.

Bilddatensätze wurden aus den folgenden Gründen eingeführt:

- Support Sie eine Vielzahl von Workflows für die medizinische Bildgebung (klinisch und nichtklinisch) über flexible APIs.

- 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?

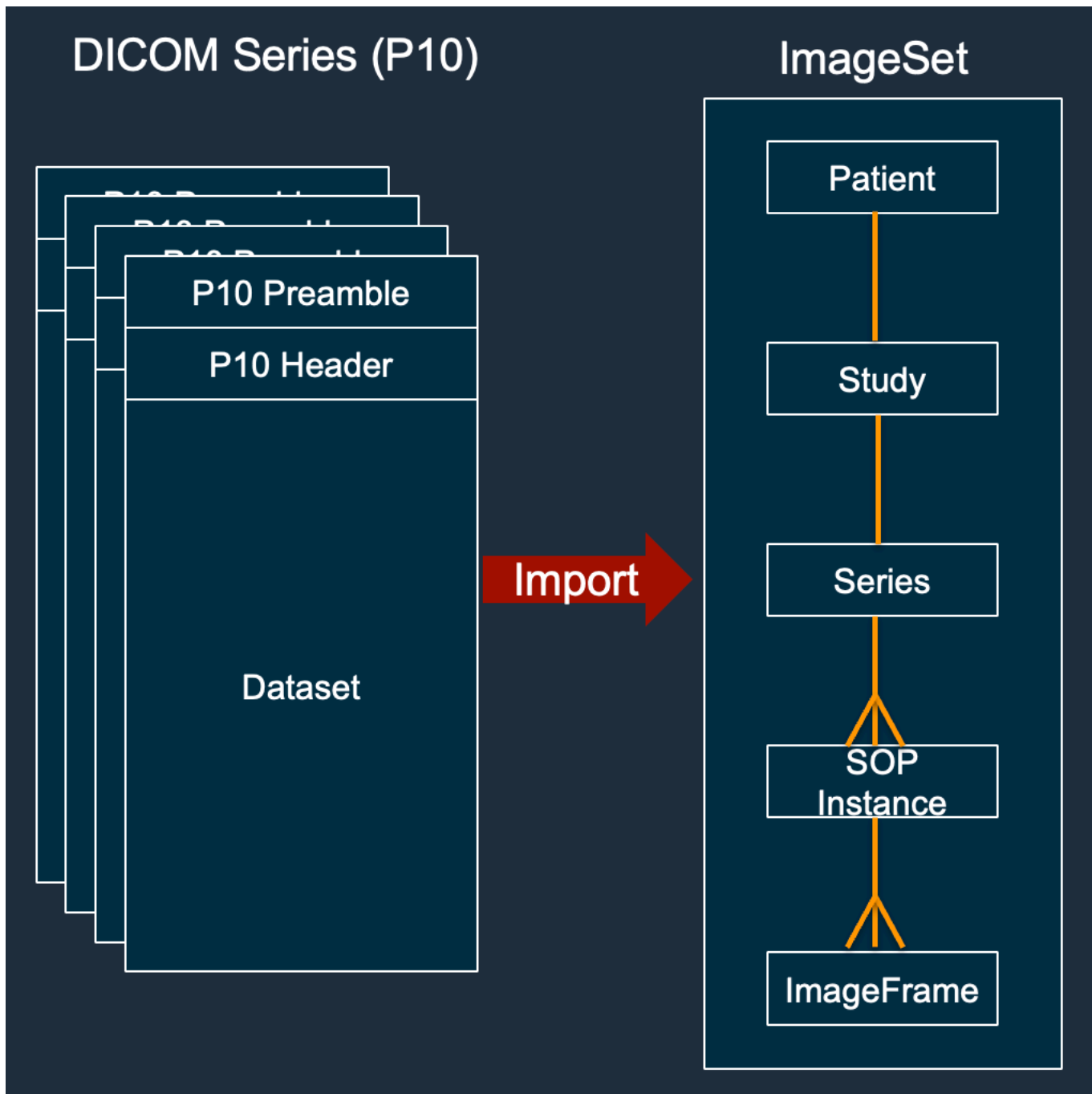
Ein Bilddatensatz ist ein AWS Konzept, das einen abstrakten Gruppierungsmechanismus zur Optimierung verwandter medizinischer Bilddaten definiert. Wenn Sie Ihre DICOM P10-Bilddaten in einen HealthImaging AWS-Datenspeicher importieren, werden sie in Bildsätze umgewandelt, die aus [Metadaten](#) und [Bildrahmen](#) (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. [Bildframes \(Pixeldaten\) sind in High-Throughput JPEG 2000 \(HTJ2K\) kodiert 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 Bilddatensätze und aktualisieren Sie niemals bestehende Bilddatensätze.

- Deduplizieren Sie den SOP-Instanzspeicher nicht, da jeder Import derselben SOP-Instanz zusätzlichen Speicherplatz beansprucht.
- Kann mehrere Bilddatensätze für eine einzelne DICOM-Serie erstellen. Zum Beispiel, wenn es eine Variante eines [normalisierten Metadatenattributs](#) gibt, z. B. eine PatientName Nichtübereinstimmung.

Wie sehen Metadaten eines Bildsatzes aus?

Verwenden Sie die `GetImageSetMetadata` Aktion, um Bildsatz-Metadaten abzurufen. Die zurückgegebenen Metadaten sind mit `komprimiertgzip`, sodass Sie sie vor der Anzeige entpacken müssen. Weitere Informationen finden Sie unter [Metadaten von Bilddatensätzen abrufen](#).

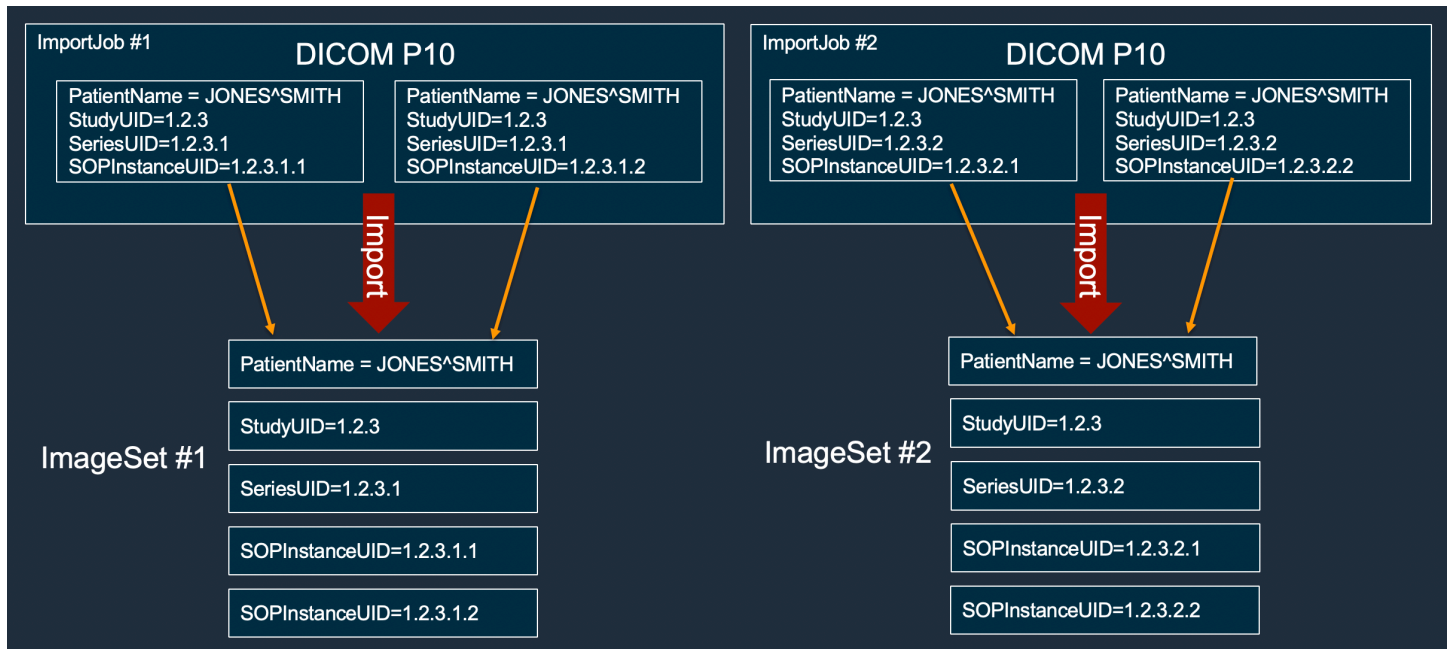
Das folgende Beispiel zeigt die Struktur von [Bilddatensatz-Metadaten](#) im JSON-Format.

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

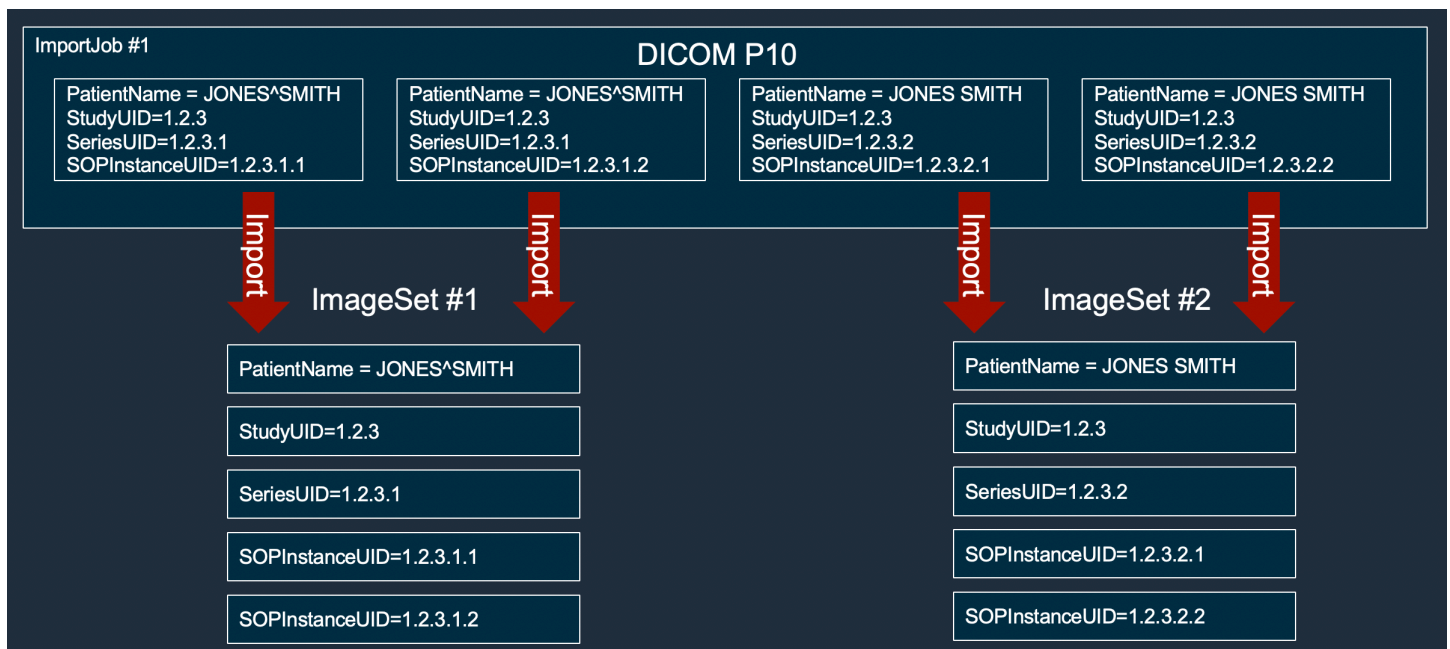
Beispiel für die Erstellung eines Bildsatzes: mehrere Importaufträge

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



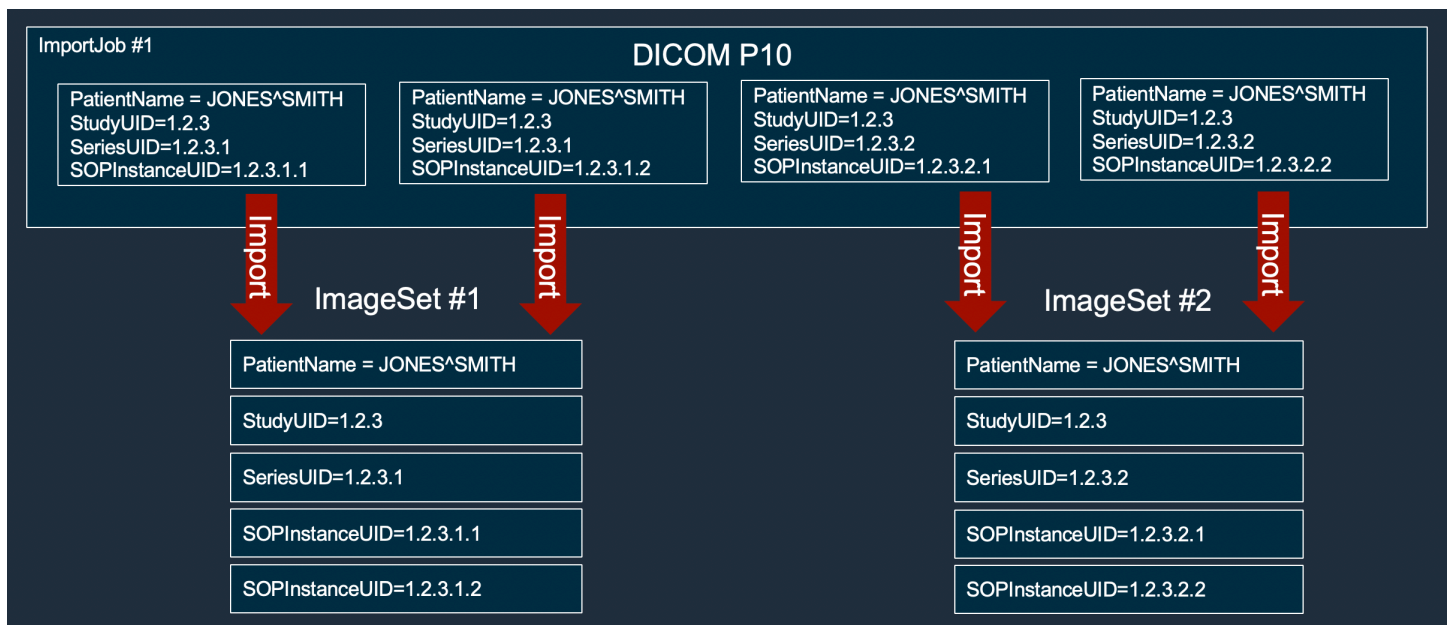
Beispiel für die Erstellung von Bilddatensätzen: Einzelner Importauftrag mit zwei Varianten

Das folgende Beispiel zeigt einen einzelnen Importauftrag, bei dem zwei Bilddatensätze erstellt werden, da die Instanzen 1 und 2 andere Patientennamen haben als die Instanzen 3 und 4.



Beispiel für die Erstellung eines Bilddatensatzes: Einzelimportauftrag 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 Suchabfragen für alle [Bilddatensätze](#) in einem ACTIVE HealthImaging Datenspeicher auszuführen. Die folgenden Menüs bieten ein Verfahren für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [SearchImageSets](#) in der AWS HealthImaging APIReferenz.

i 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 Kunden erstellt haben AWS KMS Schlüssel, Sie müssen Ihren aktualisieren AWS KMS wichtige Richtlinie vor der Interaktion mit

Bilddatensätzen. Weitere Informationen finden Sie unter [Einen vom Kunden verwalteten Schlüssel](#) erstellen.

So suchen Sie nach Bilddatensätzen

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

AWS Konsole

Note

Die folgenden Verfahren zeigen, wie Sie Bilddatensätze mithilfe der Updated at Eigenschaftsfilter Series Instance UID und durchsuchen.

Series Instance UID

Suchen Sie mit dem **Series Instance UID** Eigenschaftsfilter 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 Eigenschaftsfiltermenü und wählen Sie **Series Instance UID**.
4. Geben Sie im Feld Wert für Suche eingeben die gewünschte Serieninstanz UID ein (fügen Sie sie ein).

Note

Die UID Werte der Serieninstanz müssen mit denen identisch sein, die in der [Registry of DICOM Unique Identifiers \(UIDs\)](#) aufgeführt sind. 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 zulässigUIDs. 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 Serieninstanz UID und wählen Sie „Anwenden“.
6. Wählen Sie Search (Suchen) aus.

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

Updated at

Suchen Sie mithilfe des **Updated at** Eigenschaftensfilters 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 Eigenschaftensfiltermenü 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 Bilddatensätzen.

```
//! 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, der DICOMStudyDate und verwendetDICOMStudyTime.

```

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

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

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

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

```

```

.WithDICOMStudyDate(Aws::Utils::DateTime(std::chrono::system_clock::now()).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 verwendet createdAt. Zeitstudien wurden zuvor 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 ein DICOMSeriesInstanceUID und BETWEEN weiter updatedAt und Antwort in der ASC Reihenfolge nach updatedAt Feld sortieren.

```

    Aws::MedicalImaging::Model::SearchByAttributeValue useCase4StartDate;
    useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

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

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

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

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

    Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterEqual;

```

```

        useCase4SearchFilterEqual.SetValues({seriesInstanceUID});

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

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

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

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

        useCase4SearchCriteria.SetSort(useCase4Sort);

        Aws::Vector<Aws::String> usesCase4Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                         useCase4SearchCriteria,
                                                         usesCase4Results,
                                                         clientConfig);

        if (result) {
            std::cout << usesCase4Results.size() << " image sets found for EQUAL
operator "
                << "on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort
response\n"
                << "in ASC order on updatedAt field." << std::endl;
            for (auto &imageSetResult : usesCase4Results) {
                std::cout << " Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
    }
}

```

- API-Einheiten finden Sie [SearchImageSets](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Beispiel 1: Um Bilddatensätze mit einem EQUAL Operator zu durchsuchen

Im folgenden `search-image-sets` Codebeispiel wird der EQUAL Operator verwendet, um Bilddatensätze 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 zu durchsuchen, der DICOMStudyDate und verwendet DICOMStudyTime

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

Hinweis: DICOMStudyTime ist optional. Wenn es 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: Um Bilddatensätze mit einem BETWEEN Operator zu suchen, der createdAt (Zeitstudien wurden zuvor persistent gespeichert) verwendet

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

Hinweis: Geben Sie das Beispielformat createdAt an („1985-04-12T 23:20:50.52 Z“).

```

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 an und ein zu durchsuchen DICOMSeriesInstanceUID und die Antwort in der Reihenfolge nach dem Feld BETWEEN zu sortieren updatedAt ASC updatedAt

Im folgenden `search-image-sets` Codebeispiel wird nach Bilddatensätzen gesucht, bei denen der EQUAL Operator aktiviert DICOMSeriesInstanceUID und BETWEEN aktiviert ist, updatedAt und die Antwort wird in der ASC Reihenfolge nach dem updatedAt Feld sortiert.

Hinweis: Geben Sie das Beispielformat updatedAt an („1985-04-12T 23:20:50.52 Z“).

```

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 in der AWS HealthImaging Entwicklerhandbuch.](#)

- API-Einheiten finden Sie [SearchImageSets](#) in AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```
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, der DICOMStudyDate und verwendetDICOMStudyTime.

```

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 verwendet `createdAt`. Zeitstudien wurden zuvor 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 ein DICOMSeriesInstanceUID und BETWEEN weiter updatedAt und Antwort in der ASC Reihenfolge nach updatedAt Feld sortieren.

```
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()),
    SearchFilter.builder()
        .operator(Operator.BETWEEN)
        .values(
            SearchByAttributeValue.builder().updatedAt(startDate).build(),
            SearchByAttributeValue.builder().updatedAt(endDate).build()
        ).build());

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

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

imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
    datastoreId, searchCriteria);
if (imageSetsMetadataSummaries != null) {
```

```

        System.out.println("The image sets searched with EQUAL operator on
DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response\n" +
            "in ASC order on updatedAt field are:\n "
            + imageSetsMetadataSummaries);
        System.out.println();
    }

```

- API-Einheiten finden Sie [SearchImageSets](#) in AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```

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 DICOMStudyDate und verwendetDICOMStudyTime.

```

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 verwendet createdAt. Zeitstudien wurden zuvor 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 ein DICOMSeriesInstanceUID und BETWEEN weiter updatedAt und Antwort in der ASC Reihenfolge nach updatedAt Feld sortieren.

```
const datastoreId = "12345678901234567890123456789012";

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

```
        {
          values: [
            {DICOMSeriesInstanceUID:
"1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
            ],
          operator: "EQUAL",
        },
      ],
      sort: {
        sortOrder: "ASC",
        sortField: "updatedAt",
      }
    };

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

- API-Einheiten finden Sie [SearchImageSets](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```
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, der DICOMStudyDate und verwendetDICOMStudyTime.

```

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 verwendet createdAt. Zeitstudien wurden zuvor 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 ein DICOMSeriesInstanceUID und BETWEEN weiter updatedAt und Antwort in der ASC Reihenfolge nach updatedAt Feld sortieren.

```

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}")
```

Der folgende Code instanziiert das `MedicalImagingWrapper` Objekt.

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

- API-Einheiten finden Sie unter [SearchImageSets](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Eigenschaften von Bilddatensätzen abrufen

Verwenden Sie die `GetImageSet` Aktion, um Eigenschaften für einen bestimmten [Bilddatensatz](#) in zurückzugeben HealthImaging. Die folgenden Menüs bieten eine Vorgehensweise für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [GetImageSet](#) in der AWS HealthImaging API Referenz.

Note

AWS HealthImaging Gibt standardmäßig Eigenschaften für die neueste Version eines Bildsatzes zurück. Wenn Sie Eigenschaften für eine ältere Version eines Bildsatzes anzeigen möchten, stellen Sie dies Ihrer Anfrage `versionId` bei.

Verwenden Sie `GetDICOMInstance` HealthImaging die Darstellung eines DICOMweb Dienstes, um eine DICOM `.dcm` Instanz-Binärdatei (Datei) zurückzugeben. Weitere Informationen finden Sie unter [Eine DICOM Instanz abrufen von HealthImaging](#).

Um die Eigenschaften von Bilddatensätzen abzurufen

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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

Um die Eigenschaften von Bilddatensätzen abzurufen

Das folgende `get-image-set` Codebeispiel ruft die Eigenschaften für einen Bildsatz ab.

```
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 unter [Abrufen von Bilddatensatz-Eigenschaften](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [GetImageSet](#) in 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;  
}
```

- API-Einheiten finden Sie in [GetImageSet](#) AWS SDK for Java 2.x API Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   createdAt: 2023-09-22T14:49:26.427Z,
```



```
        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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [GetImageSetAWS SDK für Python \(Boto3\) API -Referenz](#).

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

Metadaten von Bilddatensätzen abrufen

Verwenden Sie die `GetImageSetMetadata` Aktion, um [Metadaten](#) für einen bestimmten [Bilddatensatz](#) in abzurufen HealthImaging. Die folgenden Menüs bieten ein Verfahren für AWS

Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [GetImageSetMetadata](#) in der AWS HealthImaging APIReferenz.

Note

HealthImaging Gibt standardmäßig Metadatenattribute für die neueste Version eines Bildsatzes 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 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 der Metadaten](#).

Verwenden Sie `GetDICOMInstanceMetadata` HealthImaging die Darstellung eines DICOMweb Dienstes, um DICOM Instanzmetadaten (`.jsonDatei`) zurückzugeben. Weitere Informationen finden Sie unter [DICOMInstanz-Metadaten abrufen von HealthImaging](#).

Um Metadaten eines Bildsatzes abzurufen

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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++

Utility-Funktion zum Abrufen von Bilddatensatz-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 outputPath: The path where the metadata will be stored as gzipped
 json.
 \param clientConfig: Aws client configuration.
 \\return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
&outputFilePath,
                                                  const
Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    if (!versionID.empty()) {
        request.SetVersionId(versionID);
    }
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
client.GetImageSetMetadata(
    request);
    if (outcome.IsSuccess()) {
        std::ofstream file(outputFilePath, std::ios::binary);
        auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
        file << metadata.rdbuf();
    }
    else {
        std::cerr << "Failed to get image set metadata: "
        << outcome.GetError().GetMessage() << std::endl;
    }
}
```

```
    return outcome.IsSuccess();  
}
```

Ruft Bildsatz-Metadaten ohne Version ab.

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

Holen Sie sich Bildsatz-Metadaten mit Version.

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

- API-Einheiten finden Sie [GetImageSetMetadata](#) unter AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Beispiel 1: Um Metadaten eines Bildsatzes ohne Version abzurufen

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

Hinweis: `outfile` ist ein erforderlicher Parameter

```
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 Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren. JSON

Ausgabe:

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

Beispiel 2: Um Bilddatensatz-Metadaten mit Version abzurufen

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

Hinweis: `outfile` ist ein erforderlicher Parameter

```
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 Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren. JSON

Ausgabe:

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

```
"contentEncoding": "gzip"
}
```

Weitere Informationen finden Sie unter [Abrufen von Bilddatensatz-Metadaten](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [GetImageSetMetadata](#) in 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);
    }
}
```

- API-Einheiten finden Sie in [GetImageSetMetadata](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

Utility-Funktion zum Abrufen von Bilddatensatz-Metadaten.

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

/**
 * @param {string} metadataFileName - The name of the file for the gzipped
 * metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

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

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

  console.log(response);
  // {
```

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

Ruft Bildsatz-Metadaten ohne Version ab.

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

Holen Sie sich Bildsatz-Metadaten mit Version.

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

- API-Einheiten finden Sie [GetImageSetMetadata](#) unter AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Utility-Funktion zum Abrufen von Bilddatensatz-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
```

Ruft Bildsatz-Metadaten ohne Version ab.

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

Holen Sie sich Bildsatz-Metadaten mit Version.

```
        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)
```

- API-Einheiten finden Sie unter [GetImageSetMetadata](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 HTJ2K -codierten Bildrahmen für einen bestimmten [Bilddatensatz](#) abzurufen. HealthImaging Die folgenden Menüs enthalten Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [GetImageFrame](#) in der AWS HealthImaging API Referenz.

Note

Beim [Import](#) werden alle Bildrahmen im HTJ2K verlustfreien Format AWS HealthImaging kodiert. Daher müssen sie vor der Anzeige in einem Bildbetrachter dekodiert werden. Weitere Informationen finden Sie unter [HTJ2K-Decodierungsbibliotheken](#).

`GetDICOMInstanceFrames` Verwendet HealthImaging die Darstellung eines DICOMweb Dienstes, um DICOM Instanzframes zurückzugeben (`multipartAnfrage`). Weitere Informationen finden Sie unter [DICOMInstanz-Frames abrufen von HealthImaging](#).

Um Pixeldaten eines Bilds abzurufen

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

AWS Konsole

Note

Auf Bildrahmen muss programmgesteuert zugegriffen und sie dekodiert werden, da in der Datei kein Bildbetrachter verfügbar ist AWS Management Console.

Weitere Informationen zum Dekodieren und Anzeigen von Bildrahmen finden Sie unter [HTJ2K-Decodieringsbibliotheken](#)

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();
}

```

- API-Einheiten finden Sie [GetImageFrame](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um Pixeldaten des Bildsatzes zu erhalten

Das folgende `get-image-frame` Codebeispiel ruft einen Bildrahmen ab.

```

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

```

imageframe.jpg

Hinweis: Dieses Codebeispiel beinhaltet keine Ausgabe, da die `getImageFrame` Aktion einen Stream von Pixeldaten an die Datei `imageframe.jpg` zurückgibt. Informationen zum Dekodieren und Anzeigen von Bildrahmen finden Sie unter [Bibliotheken dekodieren. HTJ2K](#)

Weitere Informationen finden Sie unter [Pixeldaten von Bilddatensätzen abrufen](#) im AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [getImageFrame](#) in 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);
    }
}
```

```
    }  
  }  
}
```

- API-Einheiten finden Sie in [GetImageFrame](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};

```

- API-Einheiten finden Sie [GetImageFrame](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [GetImageFrame](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Ändern von Bilddatensätzen mit AWS HealthImaging

DICOMBei Importaufträgen müssen Sie in der Regel Ihre [Bilddatensätze](#) aus den folgenden Gründen ändern:

- Sicherheit der Patienten
- Datenkonsistenz
- Reduzieren Sie die Lagerkosten

Wichtig

HealthImaging verarbeitet beim Import DICOM Instanz-Binärdateien (.dcmDateien) und wandelt sie in Bilddatensätze um. Verwenden Sie HealthImaging [Cloud-native Aktionen](#) (APIs), um Datenspeicher und Bildsätze zu verwalten. Verwenden Sie HealthImaging die [Darstellung von DICOMweb Diensten](#), um DICOMweb Antworten zurückzugeben.

HealthImaging bietet mehrere Cloud-native OptionenAPIs, 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 [Bildset](#) in aufzulisten HealthImaging. Die folgenden Menüs bieten eine Vorgehensweise für AWS Management Console und Codebeispiele für AWS CLI and AWS SDKs. Weitere Informationen finden Sie [ListImageSetVersions](#) in der AWS HealthImaging APIReferenz.

Note

AWS HealthImaging zeichnet jede Änderung auf, die an einem Bilddatensatz vorgenommen wurde. Durch das Aktualisieren von [Bilddatensatz-Metadaten](#) wird eine neue Version im Verlauf des Bilddatensatzes erstellt. Weitere Informationen finden Sie unter [Metadaten von Bilddatensätzen aktualisieren](#).

Um Versionen für einen Bildsatz aufzulisten

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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

Um die Versionen von Bildsätzen aufzulisten

Das folgende `list-image-set-versions` Codebeispiel listet den Versionsverlauf für einen Bildsatz auf.

```
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 unter [Auflisten von Bildsatzversionen](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [ListImageSetVersions](#) in 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;
}
```

- API-Einheiten finden Sie in [ListImageSetVersions](#) AWS SDK for Java 2.x API-Referenz.

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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

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

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   imageSetPropertiesList: [
  //     {
```

```
//          ImageSetWorkflowStatus: 'CREATED',
//          createdAt: 2023-09-22T14:49:26.427Z,
//          imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//          imageSetState: 'ACTIVE',
//          versionId: '1'
//      }]
// }
return imageSetPropertiesList;
};
```

- API-Einheiten finden Sie [ListImageSetVersions](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListImageSetVersions](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

Metadaten von Bilddatensätzen aktualisieren

Verwenden Sie die `UpdateImageSetMetadata` Aktion zum Aktualisieren von [Bildsatz-Metadaten](#) in AWS 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 auch Serien und SOP Instanzen entfernen, um Bilddatensätze mit externen Systemen synchron zu halten und Bilddatensatz-

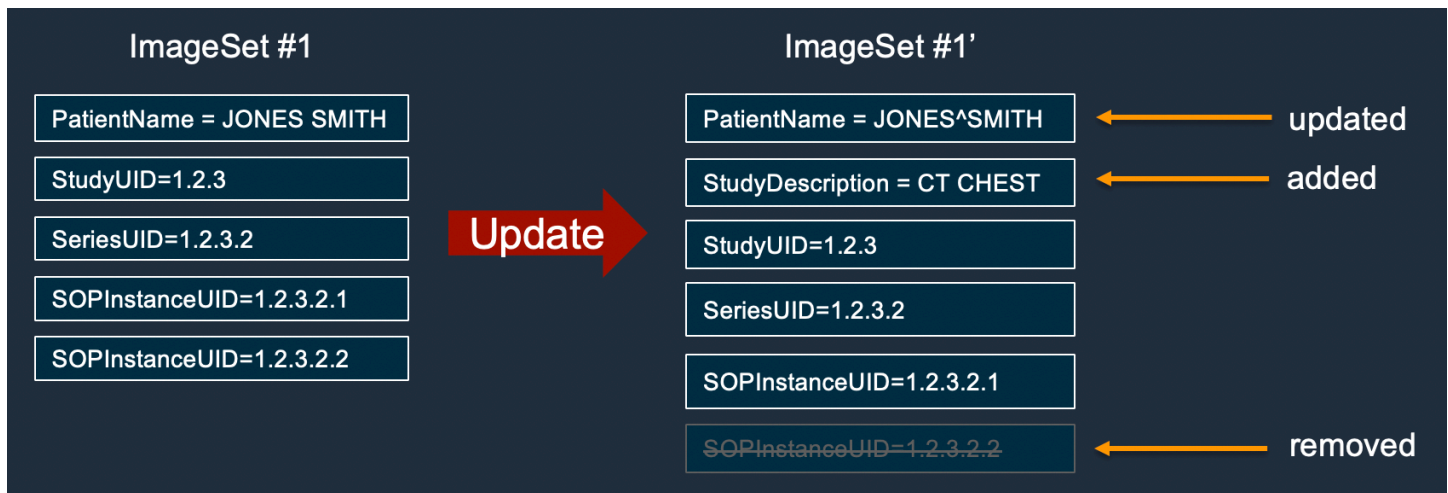
Metadaten zu anonymisieren. Weitere Informationen finden Sie [UpdateImageSetMetadata](#) in der AWS HealthImaging APIReferenz.

Note

DICOMImporte aus der realen Welt erfordern das Aktualisieren, Hinzufügen und Entfernen von Attributen aus den Metadaten des Bildsatzes. Beachten Sie bei der Aktualisierung von Bilddatensatz-Metadaten die folgenden Punkte:

- Beim 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](#) stehen [imageSetWorkflowStatus](#)Antwordelemente zur Verfügung, 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](#)Antwordelement auf und überprüfen Sie es [common errors](#).
- DICOMElementeinschränkungen werden auf Metadaten-Aktualisierungen angewendet. Mit dem [force](#)Anforderungsparameter können Sie Elemente aktualisieren, wenn Sie sie überschreiben möchten [DICOMEinschränkungen bei Metadaten](#).
- Stellen Sie den [force](#)Anforderungsparameter ein, um den Abschluss der `UpdateImageSetMetadata` Aktion zu erzwingen. Die Einstellung dieses Parameters ermöglicht die folgenden Aktualisierungen eines Bilddatensatzes:
 - Aktualisierung der `Tag.StudyID` Attribute `Tag.StudyInstanceUID` `Tag.SeriesInstanceUID` `Tag.SOPInstanceUID`,, und
 - Private DICOM Datenelemente auf Instanzebene hinzufügen, entfernen oder aktualisieren

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



Um Bildsatz-Metadaten zu aktualisieren

Wählen Sie je nach Ihren Zugriffspräferenzen eine Registerkarte für AWS HealthImaging.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: Um ein Attribut in Bildsatz-Metadaten einzufügen oder zu aktualisieren

Im folgenden `update-image-set-metadata` Beispiel wird ein Attribut in Bildsatz-Metadaten eingefügt oder aktualisiert.

```
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: Um ein Attribut aus den Metadaten eines Bildsatzes zu entfernen

Im folgenden `update-image-set-metadata` Beispiel 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: Um eine Instanz aus den Metadaten eines Bildsatzes zu entfernen

Im folgenden `update-image-set-metadata` Beispiel wird eine Instanz aus den Metadaten des 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\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}}"
  }
}

```

Ausgabe:

```

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

```

Beispiel 4: Um einen Bildsatz auf eine frühere Version zurückzusetzen

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: Um einer Instanz ein privates DICOM Datenelement hinzuzufügen

Das folgende `update-image-set-metadata` Beispiel zeigt, wie ein privates Element zu einer angegebenen Instanz innerhalb eines Bilddatensatzes hinzugefügt wird. Der DICOM Standard erlaubt private Datenelemente für die Übertragung von Informationen, die nicht in Standarddatenelementen enthalten sein können. 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
\\": {\\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\\": {\\"Instances
\\": {\\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\\": {\\"DICOM\\":
{\\"001910F9\\": \\"97\\"},\\"DICOMVRs\\": {\\"001910F9\\": \\"DS\\"}}}}}}}"
  }
}
```

Ausgabe:

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

Beispiel 6: Um ein privates DICOM Datenelement auf eine Instanz zu aktualisieren

Das folgende update-image-set-metadata Beispiel zeigt, wie der Wert eines privaten Datenelements aktualisiert wird, das zu einer Instanz innerhalb eines Bilddatensatzes gehört.

```
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
\\": {\\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\\": {\\"Instances
\\": {\\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\\": {\\"DICOM\\":
{\\"00091001\\": \\"GE_GENESIS_DD\\"}}}}}}}"
  }
}
```

```
}
}
```

Ausgabe:

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

Beispiel 7: Um a SOPInstanceUID mit dem Force-Parameter zu aktualisieren

Das folgende update-image-set-metadata Beispiel zeigt, wie eine aktualisiert wird SOPInstanceUID, 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": {
    "updateableAttributes": "{\"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 unter [Aktualisieren von Bilddatensatz-Metadaten](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [UpdateImageSetMetadata](#) in 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: Eine Instanz entfernen.

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


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 die Aktualisierung.

```
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: Eine Instanz entfernen.

```

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);
```

- API-Einheiten finden Sie unter [UpdateImageSetMetadata](#) AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

```

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: Eine Instanz entfernen.

```
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
)
```

- API-Einheiten finden Sie unter [UpdateImageSetMetadata](#) AWS SDK für Python (Boto3) API-Referenz.

Note

Es gibt noch mehr dazu. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 AWS HealthImaging 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 setzt voraus, UIDs dass die SOP Instanz innerhalb eines Bilddatensatzes eindeutig ist.
- Sie können Teilmengen von SOP Instanzen mit [copiableAttributes](#) kopieren. 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 die [message](#) Immobilie. Weitere Informationen finden Sie unter [Eigenschaften von Bilddatensätzen abrufen](#).
- DICOMImporte aus der realen Welt 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 je nach Ihren Zugriffspräferenzen eine Registerkarte für AWS HealthImaging.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: Um einen Bilddatensatz ohne Ziel zu kopieren.

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

```
aws medical-imaging copy-image-set \  
  --datastore-id 12345678901234567890123456789012 \  
  --target-image-set-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: Um einen Bilddatensatz mit einem Ziel zu kopieren.

Im folgenden `copy-image-set` Beispiel wird eine doppelte Kopie eines Bilddatensatzes mit einem 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: Um eine Teilmenge von Instanzen aus einem Quell-Bildsatz in einen Ziel-Bildsatz zu kopieren.

Im folgenden `copy-image-set` Beispiel wird eine DICOM Instanz aus dem Quell-Bildsatz in den Ziel-Bildsatz kopiert. Der `Force`-Parameter dient dazu, Inkonsistenzen in den Attributen auf Patient-, 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 unter [Kopieren eines Bilddatensatzes in](#) AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [CopyImageSet](#) in 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;
}
}

```

Utility-Funktion 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(
        ""
        ": {

```



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

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

    for (let i = 0; i < copySubsets.length; i++) {
```



```

        copySubsetsJson.Study.Series.imageSetId.Instances[
            copySubsets[i]
        ] = {};
    }

    params.copyImageSetInformation.dicomCopies = copySubsetsJson;
}

const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   datastoreId: 'xxxxxxxxxxxxxxxx',
//   destinationImageSetProperties: {
//     createdAt: 2023-09-27T19:46:21.824Z,
//     imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxx',
//     imageSetState: 'LOCKED',
//     imageSetWorkflowStatus: 'COPYING',
//     latestVersionId: '1',
//     updatedAt: 2023-09-27T19:46:21.824Z
//   },
//   sourceImageSetProperties: {
//     createdAt: 2023-09-22T14:49:26.427Z,
//     imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxx',
//     imageSetState: 'LOCKED',
//     imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//     latestVersionId: '4',
//     updatedAt: 2023-09-27T19:46:21.824Z
//   }
// }
return response;

```

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

Kopiert einen Bilddatensatz ohne Ziel.

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

Kopiert einen Bilddatensatz mit einem Ziel.

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

Kopiert eine Teilmenge eines Bilddatensatzes mit einem Ziel und erzwingt den Kopiervorgang.

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

- API-Einheiten finden Sie unter [CopyImageSet](#) AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Hilfsfunktion zum Kopieren eines Bilddatensatzes.

```
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"]

```

Kopiert einen Bilddatensatz 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,
)
```

Kopiert einen Bilddatensatz mit einem 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,
)
```

Kopiert eine Teilmenge eines Bilddatensatzes.

```
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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [CopyImageSet](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Löschen eines Bilddatensatzes

Verwenden Sie die `DeleteImageSet` Aktion, um einen [Bildsatz](#) in zu löschen HealthImaging. Die folgenden Menüs bieten eine Vorgehensweise für AWS Management Console und Codebeispiele für die AWS CLI and AWS SDKs. Weitere Informationen finden Sie [DeleteImageSet](#) in der AWS HealthImaging APIReferenz.

Um einen Bildsatz zu löschen

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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();
}

```

- API-Einheiten finden Sie [DeleteImageSet](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen Bildsatz zu löschen

Das folgende `delete-image-set` Codebeispiel löscht einen Bildsatz.

```

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 unter [Löschen eines Bilddatensatzes](#) in der AWS HealthImaging Entwicklerhandbuch.

- APIEinzelheiten finden Sie [DeleteImageSet](#) in 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);
    }
}
```

- APIEinzelheiten finden Sie in [DeleteImageSet](#) AWS SDK for Java 2.x APIReferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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

```
    return response;
};
```

- API-Einheiten finden Sie [DeleteImageSet](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [DeleteImageSet](#) AWS SDK für Python (Boto3) API - Referenz.

 Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Ressourcen taggen mit AWS HealthImaging

Sie können HealthImaging Ressourcen ([Datenspeichern](#) und [Bilddatensätzen](#)) Metadaten in Form von Tags zuweisen. Jedes Tag ist ein Label, das 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 vertrauliche Daten gedacht.

In den folgenden Themen wird beschrieben, wie HealthImaging Tagging-Operationen mithilfe von AWS Management Console, AWS CLI, und AWS SDKs verwendet werden. 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 [Bilddatensätze mit Tags](#) zu versehen AWS HealthImaging. In den folgenden Codebeispielen wird beschrieben, wie Sie die TagResource Aktion mit dem verwenden AWS Management Console, AWS CLI, und AWS SDKs. Weitere Informationen finden Sie unter [Taggen Ihres AWS Ressourcen](#) in der Allgemeine AWS-Referenz Leitfaden.

Um eine Ressource zu taggen

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: Um einen Datenspeicher zu taggen

In den folgenden tag-resource Codebeispielen wird ein Datenspeicher markiert.

```
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: Um einen Bilddatensatz zu taggen

In den folgenden tag-resource Codebeispielen wird ein Bilddatensatz markiert.

```
aws medical-imaging tag-resource \  
  --resource-arn "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012/  
imageset/18f88ac7870584f58d56256646b4d92b" \  
  --tags '{"Deployment":"Development"}'
```

```
--tags '{"Deployment":"Development"}'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Ressourcen [taggen mit AWS HealthImaging](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [TagResource](#) in 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);
    }
}
```

- API-Einheiten finden Sie in [TagResource](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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;
};
```

- API Einheiten finden Sie [TagResource](#) in AWS SDK for JavaScript API Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [TagResource](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

Tags für eine Ressource auflisten

Verwenden Sie die [ListTagsForResource](#)Aktion, um Tags für [Datenspeicher](#) und [Bilddatensätze](#) in aufzulisten AWS HealthImaging. In den folgenden Codebeispielen wird beschrieben, wie Sie die `ListTagsForResource` Aktion mit dem verwenden AWS Management Console, AWS CLI, und AWS SDKs. Weitere Informationen finden Sie unter [Taggen Ihres AWS Ressourcen](#) in der Allgemeine AWS-Referenz Leitfaden.

Um Tags für eine Ressource aufzulisten

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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: Um Ressourcen-Tags für einen Datenspeicher aufzulisten

Das folgende `list-tags-for-resource` Codebeispiel listet Tags für einen Datenspeicher auf.

```
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: Um Ressourcen-Tags für einen Bildsatz aufzulisten

Das folgende `list-tags-for-resource` Codebeispiel listet Tags für einen Bildsatz auf.

```
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](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [ListTagsForResource](#) in 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;
}

```

- API-Einheiten finden Sie in [ListTagsForResource](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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;
};
```

- API-Einheiten finden Sie [ListTagsForResource](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListTagsForResource](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel und Informationen zur Einrichtung und Ausführung finden Sie in [AWS Repository mit Codebeispielen](#).

Eine Ressource entkennzeichnen

Verwenden Sie die [UntagResource](#) Aktion, um die Markierung von [Datenspeichern](#) und [Bilddatensätzen](#) in aufzuheben. AWS HealthImaging In den folgenden Codebeispielen wird beschrieben, wie Sie die `UntagResource` Aktion mit dem verwendeten AWS Management Console, AWS CLI, und AWS SDKs. Weitere Informationen finden Sie unter [Taggen Ihres AWS Ressourcen](#) in der Allgemeine AWS-Referenz Leitfaden.

Um die Markierung einer Ressource aufzuheben

Wählen Sie ein Menü, das Ihren Zugriffspräferenzen für entspricht AWS 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.

AWS CLI und SDKs

CLI

AWS CLI

Beispiel 1: Um die Markierung eines Datenspeichers aufzuheben

Im folgenden `untag-resource` Codebeispiel wird die Markierung eines Datenspeichers aufgehoben.

```
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: Um die Markierung eines Bilddatensatzes aufzuheben

Im folgenden `untag-resource` Codebeispiel wird die Markierung eines Bilddatensatzes aufgehoben.

```
aws medical-imaging untag-resource \  
  --resource-arn "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012/  
imageset/18f88ac7870584f58d56256646b4d92b" \  
  --tag-keys ["Deployment"]
```

```
--tag-keys '["Deployment"]'
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Ressourcen [taggen mit AWS HealthImaging](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [UntagResource](#) in 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);
    }
}
```

- API-Einheiten finden Sie in [UntagResource](#) AWS SDK for Java 2.x API Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};
```

- API-Einheiten finden Sie [UntagResource](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [UntagResource](#) AWS SDK für Python (Boto3) API - Referenz.

 Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Codebeispiele für die HealthImaging Verwendung AWS SDKs

Die folgenden Codebeispiele zeigen die Verwendung HealthImaging mit einem AWS Software-Entwicklungskit (SDK).

Basics sind Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Operationen innerhalb eines Dienstes ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Aktionen zeigen Ihnen zwar, wie Sie einzelne Servicefunktionen aufrufen, aber Sie können Aktionen im Kontext der zugehörigen Szenarien sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben erledigen können, indem Sie mehrere Funktionen innerhalb eines Dienstes oder in Kombination mit anderen aufrufen AWS-Services.

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erste Schritte

Hallo HealthImaging

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung beginnen HealthImaging.

C++

SDK für C++

Code für die CMakeLists CMake .txt-Datei.

```
# 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;
}

```

- API-Einheiten finden Sie [ListDatastores](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};
```

- API-Einheiten finden Sie [ListDatastores](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Amazon HealthImaging
    client and list the data stores in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param medical_imaging_client: A Boto3 Amazon HealthImaging Client object.
    """
    print("Hello, Amazon Health Imaging! Let's list some of your data stores:\n")
    try:
        paginator = medical_imaging_client.get_paginator("list_datastores")
        page_iterator = paginator.paginate()
        datastore_summaries = []
        for page in page_iterator:
            datastore_summaries.extend(page["datastoreSummaries"])
        print("\tData Stores:")
        for ds in datastore_summaries:
            print(f"\t\tDatastore: {ds['datastoreName']} ID {ds['datastoreId']}")
    except ClientError as err:
        logger.error(
            "Couldn't list data stores. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

if __name__ == "__main__":
    hello_medical_imaging(boto3.client("medical-imaging"))
```

- API-Einheiten finden Sie in [ListDatastores](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Codebeispiele

- [Grundlegende Beispiele für die HealthImaging Verwendung AWS SDKs](#)

- [Hallo HealthImaging](#)
- [Aktionen zur HealthImaging Verwendung AWS SDKs](#)
 - [Verwenden Sie CopyImageSet mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie CreateDatastore mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie DeleteDatastore mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie DeletelImageSet mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetDICOMImportJob mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetDatastore mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetImageFrame mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetImageSet mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetImageSetMetadata mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListDICOMImportJobs mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListDatastores mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListImageSetVersions mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListTagsForResource mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie SearchImageSets mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie StartDICOMImportJob mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie TagResource mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie UntagResource mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie UpdateImageSetMetadata mit einem AWS SDKoder CLI](#)
- [Szenarien für die HealthImaging Verwendung AWS SDKs](#)
 - [Beginnen Sie mit HealthImaging Bilddatensätzen und Bildrahmen mithilfe eines AWS SDK](#)
 - [Kennzeichnen eines HealthImaging Datenspeichers mit einem AWS SDK](#)
 - [Markieren eines HealthImaging Bilddatensatzes mit einem AWS SDK](#)

Grundlegende Beispiele für die HealthImaging Verwendung AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie die Grundlagen von verwenden AWS HealthImaging mit AWS SDKs.

Beispiele

- [Hallo HealthImaging](#)
- [Aktionen zur HealthImaging Verwendung AWS SDKs](#)
 - [Verwenden Sie CopyImageSet mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie CreateDatastore mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie DeleteDatastore mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie DeletelImageSet mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetDICOMImportJob mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetDatastore mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetImageFrame mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetImageSet mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie GetImageSetMetadata mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListDICOMImportJobs mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListDatastores mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListImageSetVersions mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie ListTagsForResource mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie SearchImageSets mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie StartDICOMImportJob mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie TagResource mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie UntagResource mit einem AWS SDKoder CLI](#)
 - [Verwenden Sie UpdateImageSetMetadata mit einem AWS SDKoder CLI](#)

Hallo HealthImaging

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung beginnen HealthImaging.

C++

SDKfür C++

Code für die CMakeLists CMake .txt-Datei.

```
# 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;
}

```

- APIEinzelheiten finden Sie [ListDatastores](#) in AWS SDK for C++ APIReferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};
```

- API Einheiten finden Sie [ListDatastores](#) in AWS SDK for JavaScript API Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Amazon HealthImaging
    client and list the data stores in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param medical_imaging_client: A Boto3 Amazon HealthImaging Client object.
    """
    print("Hello, Amazon Health Imaging! Let's list some of your data stores:\n")
    try:
        paginator = medical_imaging_client.get_paginator("list_datastores")
        page_iterator = paginator.paginate()
        datastore_summaries = []
        for page in page_iterator:
            datastore_summaries.extend(page["datastoreSummaries"])
        print("\tData Stores:")
        for ds in datastore_summaries:
            print(f"\t\tDatastore: {ds['datastoreName']} ID {ds['datastoreId']}")
    except ClientError as err:
        logger.error(
            "Couldn't list data stores. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

if __name__ == "__main__":
    hello_medical_imaging(boto3.client("medical-imaging"))
```


- API-Einheiten finden Sie in [List Datastores AWS SDK für Python \(Boto3\) API -Referenz](#).

Note

Es gibt noch mehr dazu. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging 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 AWS SDKs ausführen. Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes finden.

Bei diesen Auszügen handelt es sich um Codeauszüge aus größeren Programmen, die im Kontext ausgeführt werden müssen. Aktionen im Kontext finden Sie unter [Szenarien für die HealthImaging-Verwendung AWS SDKs](#).

Die folgenden Beispiele enthalten nur die am häufigsten verwendeten Aktionen. Eine vollständige Liste finden Sie im [AWS HealthImaging API-Referenz](#).

Beispiele

- [Verwenden Sie CopyImageSet mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie CreateDatastore mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie DeleteDatastore mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie DeleteImageSet mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie GetDICOMImportJob mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie GetDatastore mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie GetImageFrame mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie GetImageSet mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie GetImageSetMetadata mit einem AWS SDK-Koder CLI](#)
- [Verwenden Sie ListDICOMImportJobs mit einem AWS SDK-Koder CLI](#)

- [Verwenden Sie ListDatastores mit einem AWS SDKoder CLI](#)
- [Verwenden Sie ListImageSetVersions mit einem AWS SDKoder CLI](#)
- [Verwenden Sie ListTagsForResource mit einem AWS SDKoder CLI](#)
- [Verwenden Sie SearchImageSets mit einem AWS SDKoder CLI](#)
- [Verwenden Sie StartDICOMImportJob mit einem AWS SDKoder CLI](#)
- [Verwenden Sie TagResource mit einem AWS SDKoder CLI](#)
- [Verwenden Sie UntagResource mit einem AWS SDKoder CLI](#)
- [Verwenden Sie UpdateImageSetMetadata mit einem AWS SDKoder CLI](#)

Verwenden Sie **CopyImageSet** mit einem AWS SDKoder CLI

Die folgenden Codebeispiele zeigen, wie man es benutztCopyImageSet.

CLI

AWS CLI

Beispiel 1: Um einen Bilddatensatz ohne Ziel zu kopieren.

Im folgenden copy-image-set Beispiel wird eine doppelte Kopie eines Bilddatensatzes 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: Um einen Bilddatensatz mit einem Ziel zu kopieren.

Im folgenden `copy-image-set` Beispiel wird eine doppelte Kopie eines Bilddatensatzes mit einem 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: Um eine Teilmenge von Instanzen aus einem Quell-Bildsatz in einen Ziel-Bildsatz zu kopieren.

Im folgenden `copy-image-set` Beispiel wird eine DICOM Instanz aus dem Quell-Bildsatz in den Ziel-Bildsatz kopiert. Der `Force`-Parameter dient dazu, Inkonsistenzen in den Attributen auf Patient-, 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 unter [Kopieren eines Bilddatensatzes in AWS HealthImaging](#) Entwicklerhandbuch.

- API-Einheiten finden Sie [CopyImageSet](#) in 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;
    }
}
```

Utility-Funktion 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();
}

```

- API-Einheiten finden Sie unter [CopyImageSet](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

Utility-Funktion zum Kopieren eines Bilddatensatzes.

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination
 image set.
 * @param {string} destinationVersionId - The optional version ID of the
 destination image set.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = []
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
```



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

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

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

    params.copyImageSetInformation.dicomCopies = copySubsetsJson;
}

const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
);
console.log(response);
// {
//     '$metadata': {
//         httpStatusCode: 200,
//         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
//         extendedRequestId: undefined,
//         cfId: undefined,
```

```

        //      attempts: 1,
        //      totalRetryDelay: 0
        //    },
        //    datastoreId: 'xxxxxxxxxxxxxxxx',
        //    destinationImageSetProperties: {
        //      createdAt: 2023-09-27T19:46:21.824Z,
        //      imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxx',
        //      imageSetId: 'xxxxxxxxxxxxxxxxxxxxx',
        //      imageSetState: 'LOCKED',
        //      imageSetWorkflowStatus: 'COPYING',
        //      latestVersionId: '1',
        //      updatedAt: 2023-09-27T19:46:21.824Z
        //    },
        //    sourceImageSetProperties: {
        //      createdAt: 2023-09-22T14:49:26.427Z,
        //      imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxx',
        //      imageSetId: 'xxxxxxxxxxxxxxxxxxxxx',
        //      imageSetState: 'LOCKED',
        //      imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
        //      latestVersionId: '4',
        //      updatedAt: 2023-09-27T19:46:21.824Z
        //    }
        //  }
    return response;
} catch (err) {
    console.error(err);
}
};

```

Kopiert einen Bilddatensatz ohne Ziel.

```

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

```

Kopiert einen Bilddatensatz mit einem Ziel.

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

Kopiert eine Teilmenge eines Bilddatensatzes mit einem Ziel und erzwingt den Kopiervorgang.

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

- API-Einheiten finden Sie unter [CopyImageSet](#) AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu [GitHub](#). Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Hilfsfunktion zum Kopieren eines Bilddatensatzes.

```
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"]

```

Kopiert einen Bilddatensatz 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,
)

```

Kopiert einen Bilddatensatz mit einem 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,
)

```

Kopiert eine Teilmenge eines Bilddatensatzes.

```

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 Objekt. `MedicalImagingWrapper`

```

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

```

- API-Einheiten finden Sie unter [CopyImageSet](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. [GitHub](#) Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **CreateDatastore** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `CreateDatastore`.

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 getopt command in a function.

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

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

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

```



```
usage
return 1
fi

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

local error_code=${?}

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

echo "$response"

return 0
}
```

- API-Einheiten finden Sie in [CreateDatastore](#) AWS CLI Befehlsreferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen Datenspeicher zu erstellen

Das folgende `create-datastore` Codebeispiel erstellt einen Datenspeicher mit dem Namen `my-datastore`.

```
aws medical-imaging create-datastore \
```

```
--datastore-name "my-datastore"
```

Ausgabe:

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

Weitere Informationen finden Sie unter [Erstellen eines Datenspeichers](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [CreateDatastore](#) in 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 "";
}
```

- API-Einheiten finden Sie in [CreateDatastore](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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

- API-Einheiten finden Sie [CreateDatastore](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [CreateDatastore](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **DeleteDatastore** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `DeleteDatastore`.

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 getopt command in a function.

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

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

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

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

    local error_code=${?}

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

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

return 0
}
```

- API-Einheiten finden Sie in [DeleteDatastore](#) AWS CLI Befehlsreferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen Datenspeicher zu löschen

Das folgende `delete-datastore` Codebeispiel löscht einen Datenspeicher.

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

Ausgabe:

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

Weitere Informationen finden Sie unter [Löschen eines Datenspeichers](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [DeleteDatastore](#) in 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);
    }
}
```

- API-Einheiten finden Sie in [DeleteDatastore](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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

return response;
};

```

- API-Einheiten finden Sie [DeleteDatastore](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [DeleteDatastore](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **DeleteImageSet** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `DeleteImageSet`.

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 Bildrahmen](#)


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();
}
```

- API-Einheiten finden Sie [DeleteImageSet](#) in AWS SDK for C++ API-Referenz.

 Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen Bildsatz zu löschen

Das folgende `delete-image-set` Codebeispiel löscht einen Bildsatz.

```
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 unter [Löschen eines Bilddatensatzes](#) in AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [DeleteImageSet](#) in 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);
    }
}

```

- API-Einheiten finden Sie in [DeleteImageSet](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 = "xxxxxxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,

```

```
//      requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    datastoreId: 'xxxxxxxxxxxxxxxxxxxx',
//    imageSetId: 'xxxxxxxxxxxxxxxxxxxx',
//    imageSetState: 'LOCKED',
//    imageSetWorkflowStatus: 'DELETING'
// }
return response;
};
```

- API-Einheiten finden Sie [DeleteImageSet](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [DeletImageSet](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **GetDICOMImportJob** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `GetDICOMImportJob`.

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 Bildrahmen](#)

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;
}
```

- API-Einheiten finden Sie unter [GetDICOMImportJob](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um die Eigenschaften eines DICOM-Importjobs abzurufen

Im folgenden `get-dicom-import-job` Codebeispiel 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 unter [Abrufen der Eigenschaften eines Importauftrags](#) im AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie unter [GetDICOMImport Job](#) in 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;
}
```

- API-Einheiten finden Sie unter [GetDICOMImportJob](#) in AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 = "xxxxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a2637936-78ea-44e7-98b8-7a87d95dface',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   jobProperties: {
  //     dataAccessRoleArn: 'arn:aws:iam:xxxxxxxxxxxx:role/dicom_import',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //     endedAt: 2023-09-19T17:29:21.753Z,
  //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobName: 'job_1',
  //     jobStatus: 'COMPLETED',
  //     outputS3Uri: 's3://health-imaging-dest/
  output_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxx'/',
  //     submittedAt: 2023-09-19T17:27:25.143Z
  //   }
  // }

  return response;
};

```

- APIEinzelheiten finden Sie unter [GetDICOMImport Job](#) in AWS SDK for JavaScript APIReferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

```
medical_imaging_wrapper = MedicalImagingWrapper(client)
```

- API-Einheiten finden Sie unter [GetDICOMImport Job](#) in AWS SDK für Python (Boto3) API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **GetDatastore** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `GetDatastore`.

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 getopt command in a
    function.
    local error_code
    # bashsupport disable=BP5008
    function usage() {
        echo "function imaging_get_datastore"
        echo "Gets a data store's properties."
        echo "  -i datastore_id - The ID of the data store."
        echo ""
    }

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

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

    local response

    response=$(

```

```
aws medical-imaging get-datastore \  
  --datastore-id "$datastore_id" \  
  --output text \  
  --query "[ datastoreProperties.datastoreName,  
datastoreProperties.datastoreId, datastoreProperties.datastoreStatus,  
datastoreProperties.datastoreArn,  datastoreProperties.createdAt,  
datastoreProperties.updatedAt]"  
)  
error_code=${?}  
  
if [[ $error_code -ne 0 ]]; then  
  aws_cli_error_log $error_code  
  errecho "ERROR: AWS reports list-datastores operation failed.$response"  
  return 1  
fi  
  
echo "$response"  
  
return 0  
}
```

- API-Einheiten finden Sie in [GetDatastore AWS CLI Befehlsreferenz](#).

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um die Eigenschaften eines Datenspeichers abzurufen

Das folgende `get-datastore` Codebeispiel ruft die Eigenschaften eines Datenspeichers ab.

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

Ausgabe:

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

Weitere Informationen finden Sie unter [Abrufen der Eigenschaften eines Datenspeichers](#) in AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [GetDatastore](#) in 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;
}
```

- API-Einheiten finden Sie in [GetDatastore](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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

- API-Einheiten finden Sie [GetDatastore](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [GetDatastore](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **GetImageFrame** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `GetImageFrame`.

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 Bildrahmen](#)

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();
}
```

- API-Einheiten finden Sie [GetImageFrame](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um Pixeldaten des Bildsatzes zu erhalten

Das folgende `get-image-frame` Codebeispiel ruft einen Bildrahmen ab.

```
aws medical-imaging get-image-frame \  
  --datastore-id "12345678901234567890123456789012" \  
  --image-set-id "98765412345612345678907890789012" \  
  --image-frame-information imageFrameId=3abf5d5d7ae72f80aec81b2c0de3ef4 \  
  imageframe.jpg
```

Hinweis: Dieses Codebeispiel beinhaltet keine Ausgabe, da die `GetImageFrame` Aktion einen Stream von Pixeldaten an die Datei `imageframe.jpg` zurückgibt. Informationen zum Dekodieren und Anzeigen von Bildrahmen finden Sie unter Bibliotheken `dekodieren`. HTJ2K

Weitere Informationen finden Sie unter [Pixeldaten von Bilddatensätzen abrufen](#) im AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [GetImageFrame](#) in 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);
    }
}

```

- API-Einheiten finden Sie in [GetImageFrame AWS SDK for Java 2.x API-Referenz](#).

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};
```

- API-Einheiten finden Sie [GetImageFrame](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [GetImageFrame](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **GetImageSet** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `GetImageSet`.

CLI

AWS CLI

Um die Eigenschaften von Bilddatensätzen abzurufen

Das folgende `get-image-set` Codebeispiel ruft die Eigenschaften für einen Bildsatz ab.

```
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 unter [Abrufen von Bilddatensatz-Eigenschaften](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [GetImageSet](#) in 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;
  }
```

- API-Einheiten finden Sie in [GetImageSetAWS SDK for Java 2.x API-Referenz](#).

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```



```
: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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [GetImageSet](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **GetImageSetMetadata** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `GetImageSetMetadata`.

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 Bildrahmen](#)

C++

SDK für C++

Utility-Funktion zum Abrufen von Bilddatensatz-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 outputPath: The path where the metadata will be stored as gzipped
  json.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
&outputFilePath,
                                                  const
Aws::Client::ClientConfiguration &clientConfig) {
  Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
  request.SetDatastoreId(dataStoreID);
  request.SetImageSetId(imageSetID);
  if (!versionID.empty()) {
    request.SetVersionId(versionID);
  }
}
```

```

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

    return outcome.IsSuccess();
}

```

Ruft 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;
    }

```

Holen Sie sich Bildsatz-Metadaten mit Version.

```

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

```

- API-Einheiten finden Sie [GetImageSetMetadata](#) unter AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Beispiel 1: Um Metadaten eines Bildsatzes ohne Version abzurufen

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

Hinweis: `outfile` ist ein erforderlicher Parameter

```
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 Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren. JSON

Ausgabe:

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

Beispiel 2: Um Bilddatensatz-Metadaten mit Version abzurufen

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

Hinweis: `outfile` ist ein erforderlicher Parameter

```
aws medical-imaging get-image-set-metadata \  
  --datastore-id 12345678901234567890123456789012 \  
  --image-set-id ea92b0d8838c72a3f25d00d13616f87e \  
  --version 1.0.0 \  
  studymetadata.json.gz
```



```
--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 Objekts anzuzeigen, müssen Sie es zuerst dekomprimieren. JSON

Ausgabe:

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

Weitere Informationen finden Sie unter [Abrufen von Bilddatensatz-Metadaten](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [GetImageSetMetadata](#) in 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);
}
}

```

- API-Einheiten finden Sie in [GetImageSetMetadata](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

Utility-Funktion zum Abrufen von Bilddatensatz-Metadaten.

```

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

/**
 * @param {string} metadataFileName - The name of the file for the gzipped
 * metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
    metadataFileName = "metadata.json.gz",
    datastoreId = "xxxxxxxxxxxxxxxx",
    imagesetId = "xxxxxxxxxxxxxxxx",
    versionID = ""

```

```

) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

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

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

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

  return response;
};

```

Ruft Bildsatz-Metadaten ohne Version ab.

```

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

```

```
}
```

Holen Sie sich Bildsatz-Metadaten mit Version.

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

- API-Einheiten finden Sie [getImageSetMetadata](#) unter AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Utility-Funktion zum Abrufen von Bilddatensatz-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
```

Ruft Bildsatz-Metadaten ohne Version ab.

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

```
)
```

Holen Sie sich Bildsatz-Metadaten mit Version.

```
        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)
```

- API-Einheiten finden Sie unter [GetImageSetMetadata](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **ListDICOMImportJobs** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `ListDICOMImportJobs`.

CLI

AWS CLI

Um DICOM-Importaufträge aufzulisten

Das folgende `list-dicom-import-jobs` Codebeispiel listet DICOM-Importaufträge auf.

```
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 Leitfaden für Entwickler.

- Weitere API Informationen finden Sie unter [listDICOMImportL-Jobs](#) in 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<>();
}

```

- APIEinzelheiten finden Sie unter [ListDICOMImport Jobs in AWS SDK for Java 2.x APIReferenz](#).

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 = "xxxxxxxxxxxxxxxxxxxxxx"
) => {
    const paginatorConfig = {
        client: medicalImagingClient,
        pageSize: 50,
    };

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

```



```

let jobSummaries = [];
for await (const page of paginator) {
  // Each page contains a list of `jobSummaries`. The list is truncated if is
  // larger than `pageSize`.
  jobSummaries.push(...page["jobSummaries"]);
  console.log(page);
}
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   jobSummaries: [
//     {
//       dataAccessRoleArn: 'arn:aws:iam:xxxxxxxxxxxx:role/
dicom_import',
//       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       endedAt: 2023-09-22T14:49:51.351Z,
//       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       jobName: 'test-1',
//       jobStatus: 'COMPLETED',
//       submittedAt: 2023-09-22T14:48:45.767Z
//     }
//   ]
}

return jobSummaries;
};

```

- API-Einheiten finden Sie unter [ListDICOMImportJobs](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListDICOMImport](#) Jobs in AWS SDK für Python (Boto3) API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **ListDatastores** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `ListDatastores`.

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 getopt command in a function.
local error_code
# bashsupport disable=BP5008
function usage() {
    echo "function imaging_list_datastores"
    echo "Lists the AWS HealthImaging data stores in the account."
    echo ""
}

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

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

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

echo "$response"

return 0
}
```

- API-Einheiten finden Sie in [ListDatastores](#) AWS CLI Befehlsreferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um Datenspeicher aufzulisten

Das folgende `list-datastores` Codebeispiel listet die verfügbaren Datenspeicher auf.

```
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 unter [Auflisten von Datenspeichern](#) im AWS HealthImaging Leitfadens für Entwickler.

- API-Einheiten finden Sie [ListDatastores](#) in 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;
}
```

- API-Einheiten finden Sie in [ListDatastores](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

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


- API-Einheiten finden Sie [ListDatastores](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListDatastores](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`ListImageSetVersions`** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `ListImageSetVersions`.

CLI

AWS CLI

Um Versionen von Bildsätzen aufzulisten

Das folgende `list-image-set-versions` Codebeispiel listet den Versionsverlauf für einen Bildsatz auf.

```
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 unter [Auflisten von Bildsatzversionen](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [ListImageSetVersions](#) in 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;
}
```

- API-Einheiten finden Sie in [ListImageSetVersions](#) AWS SDK for Java 2.x API-Referenz.

 Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   imageSetPropertiesList: [
  //     {
```

```
//      ImageSetWorkflowStatus: 'CREATED',
//      createdAt: 2023-09-22T14:49:26.427Z,
//      imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      imageSetState: 'ACTIVE',
//      versionId: '1'
//    }]
// }
return imageSetPropertiesList;
};
```

- API-Einheiten finden Sie [ListImageSetVersions](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListImageSetVersions](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **ListTagsForResource** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `ListTagsForResource`.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Kennzeichen eines Datenspeichers](#)
- [Einen Bilddatensatz taggen](#)

CLI

AWS CLI

Beispiel 1: Um Ressourcen-Tags für einen Datenspeicher aufzulisten

Das folgende `list-tags-for-resource` Codebeispiel listet Tags für einen Datenspeicher auf.

```
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: Um Ressourcen-Tags für einen Bildsatz aufzulisten

Das folgende `list-tags-for-resource` Codebeispiel listet Tags für einen Bildsatz auf.

```
aws medical-imaging list-tags-for-resource \  
  --resource-arn "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012/  
image/18f88ac7870584f58d56256646b4d92b"
```

Ausgabe:

```
{  
  "tags":{  
    "Deployment":"Development"  
  }  
}
```

Weitere Informationen finden Sie unter Ressourcen [taggen mit AWS HealthImaging](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [ListTagsForResource](#) in 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;
}
```

- API-Einheiten finden Sie in [ListTagsForResource](#) AWS SDK for Java 2.x API Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};
```

- API-Einheiten finden Sie [ListTagsForResource](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [ListTagsForResource](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDKEntwicklerhandbücher und Codebeispiele finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK Versionen.

Verwenden Sie **SearchImageSets** mit einem AWS SDKoder CLI

Die folgenden Codebeispiele zeigen, wie man es benutztSearchImageSets.

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 Bildrahmen](#)

C++

SDKfür C++

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```
//! 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(patientID)});

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

searchCriteriaEqualsPatientID,

imageIDsForPatientID,

clientConfig);

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

```

Anwendungsfall #2: BETWEEN Operator verwendet DICOMStudyDate undDICOMStudyTime.

```

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

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

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

useCase2EndDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAndTime()
.WithDICOMStudyDate(Aws::Utils::DateTime(std::chrono::system_clock::now()).ToLocalTimeStr("%m%d"))
.WithDICOMStudyTime("000000.000"));

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

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

```

```

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

Aws::Vector<Aws::String> usesCase2Results;
result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                    useCase2SearchCriteria,
                                                    usesCase2Results,
                                                    clientConfig);

if (result) {
    std::cout << usesCase2Results.size() << " image sets found for
between 1999/01/01 and present."
              << std::endl;
    for (auto &imageSetResult : usesCase2Results) {
        std::cout << " Image set with ID '" << imageSetResult <<
std::endl;
    }
}

```

Anwendungsfall #3: BETWEEN Operator verwendet
 createdAt. Zeitstudien wurden zuvor 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 ein DICOMSeriesInstanceUID und BETWEEN weiter updatedAt und Antwort in der ASC Reihenfolge nach updatedAt Feld sortieren.

```

    Aws::MedicalImaging::Model::SearchByAttributeValue useCase4StartDate;
useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

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

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

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

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

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

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

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

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

```

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

useCase4SearchCriteria.SetSort(useCase4Sort);

Aws::Vector<Aws::String> usesCase4Results;
result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                    useCase4SearchCriteria,
                                                    usesCase4Results,
                                                    clientConfig);

if (result) {
    std::cout << usesCase4Results.size() << " image sets found for EQUAL
operator "
    << "on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort
response\n"
    << "in ASC order on updatedAt field." << std::endl;
    for (auto &imageSetResult : usesCase4Results) {
        std::cout << " Image set with ID '" << imageSetResult <<
std::endl;
    }
}
```

- API Einheiten finden Sie [SearchImageSets](#) in AWS SDK for C++ API Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Beispiel 1: Um Bilddatensätze mit einem EQUAL Operator zu durchsuchen

Im folgenden `search-image-sets` Codebeispiel wird der EQUAL Operator verwendet, um Bilddatensätze 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 zu durchsuchen, der DICOMStudyDate und verwendet DICOMStudyTime

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

Hinweis: DICOMStudyTime ist optional. Wenn es 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",  
    }  
  }  
}
```

```

        "DICOPatientName": "Melissa844 Huel628",
        "DICOMNumberOfStudyRelatedInstances": 1,
        "DICOMStudyTime": "140728",
        "DICOMNumberOfStudyRelatedSeries": 1
    },
    "updatedAt": "2022-12-06T21:40:59.429000+00:00"
  ]]
}

```

Beispiel 3: Um Bilddatensätze mit einem BETWEEN Operator zu suchen, der createdAt (Zeitstudien wurden zuvor persistent gespeichert) verwendet

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

Hinweis: Geben Sie das Beispielformat `createdAt` an („1985-04-12T 23:20:50.52 Z“).

```

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 an und ein zu durchsuchen DICOMSeriesInstanceUID und die Antwort in der Reihenfolge nach dem Feld BETWEEN zu sortieren updatedAt ASC updatedAt

Im folgenden `search-image-sets` Codebeispiel wird nach Bilddatensätzen gesucht, bei denen der EQUAL Operator aktiviert DICOMSeriesInstanceUID und BETWEEN aktiviert ist, updatedAt und die Antwort wird in der ASC Reihenfolge nach dem updatedAt Feld sortiert.

Hinweis: Geben Sie das Beispielformat updatedAt an („1985-04-12T 23:20:50.52 Z“).

```

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 in der AWS HealthImaging Entwicklerhandbuch.](#)

- API-Einheiten finden Sie [SearchImageSets](#) in AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```
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 verwendet DICOMStudyDate und DICOMStudyTime.

```

        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 verwendet `createdAt`. Zeitstudien wurden zuvor 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 ein `DICOMSeriesInstanceUID` und BETWEEN weiter `updatedAt` und Antwort in der ASC Reihenfolge nach `updatedAt` Feld sortieren.


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

searchFilters = Arrays.asList(
    SearchFilter.builder()
        .operator(Operator.EQUAL)
        .values(SearchByAttributeValue.builder()
            .dicomSeriesInstanceUID(seriesInstanceUID)
            .build())
        .build(),
    SearchFilter.builder()
        .operator(Operator.BETWEEN)
        .values(
            SearchByAttributeValue.builder().updatedAt(startDate).build(),
            SearchByAttributeValue.builder().updatedAt(endDate).build()
        ).build());

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

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

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

- API-Einheiten finden Sie [SearchImageSets](#) in AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```
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 verwendet DICOMStudyDate und DICOMStudyTime.

```
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 verwendet createdAt. Zeitstudien wurden zuvor 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 ein DICOMSeriesInstanceUID und BETWEEN weiter updatedAt und Antwort in der ASC Reihenfolge nach updatedAt Feld sortieren.

```

const datastoreId = "12345678901234567890123456789012";

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

```

```

};

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

```

- API-Einheiten finden Sie [SearchImageSets](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```

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 verwendet DICOMStudyDate und DICOMStudyTime.

```

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 verwendet createdAt. Zeitstudien wurden zuvor 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 ein DICOMSeriesInstanceUID und BETWEEN weiter updatedAt und Antwort in der ASC Reihenfolge nach updatedAt Feld sortieren.

```

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}")

```

Der folgende Code instanziiert das MedicalImagingWrapper Objekt.

```

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

```

- API-Einheiten finden Sie unter [SearchImageSets](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **StartDICOMImportJob** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `StartDICOMImportJob`.

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 Bildrahmen](#)

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();
}

```

- API-Einheiten finden Sie unter [StartDICOMImport Job](#) in AWS SDK for C++ API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

CLI

AWS CLI

Um einen DICOM-Importjob zu starten

Das folgende `start-dicom-import-job` Codebeispiel startet einen DICOM-Importauftrag.

```
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 Leitfaden für Entwickler.

- API-Einheiten finden Sie unter [StartDICOMImport Job](#) in 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 "";
}
```

- API-Einheiten finden Sie unter [StartDICOMImportJob](#) in AWS SDK for Java 2.x API-Referenz.

 Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',

```

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

- APIEinzelheiten finden Sie unter [StartDICOMImport Job](#) in AWS SDK for JavaScript APIReferenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [StartDICOMImport Job](#) in AWS SDK für Python (Boto3) API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **TagResource** mit einem AWS SDK oder CLI

Die folgenden Codebeispiele zeigen, wie man es benutzt `TagResource`.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Kennzeichen eines Datenspeichers](#)
- [Einen Bilddatensatz taggen](#)

CLI

AWS CLI

Beispiel 1: Um einen Datenspeicher zu taggen

Die folgenden `tag-resource` Codebeispiele kennzeichnen einen Datenspeicher.

```
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: Um einen Bilddatensatz zu taggen

In den folgenden `tag-resource` Codebeispielen wird ein Bilddatensatz markiert.

```
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](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [TagResource](#) in 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);
    }
}
```

- API-Einheiten finden Sie in [TagResource](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};

```

- API-Einheiten finden Sie [TagResource](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [TagResource](#) AWS SDK für Python (Boto3) API -Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **UntagResource** mit einem AWS SDKoder CLI

Die folgenden Codebeispiele zeigen, wie man es benutztUntagResource.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Kennzeichnen eines Datenspeichers](#)
- [Einen Bilddatensatz taggen](#)

CLI

AWS CLI

Beispiel 1: Um die Markierung eines Datenspeichers aufzuheben

Im folgenden `untag-resource` Codebeispiel wird die Markierung eines Datenspeichers aufgehoben.

```
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: Um die Markierung eines Bilddatensatzes aufzuheben

Im folgenden `untag-resource` Codebeispiel wird die Markierung eines Bilddatensatzes aufgehoben.

```
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](#) in der AWS HealthImaging Leitfaden für Entwickler.

- API-Einheiten finden Sie [UntagResource](#) in 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);
    }
}
```

- API-Einheiten finden Sie in [UntagResource](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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;
};
```

- API-Einheiten finden Sie [UntagResource](#) in AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie unter [UntagResource](#) AWS SDK für Python (Boto3) API - Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDKEntwicklerhandbücher und Codebeispiele finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK Versionen.

Verwenden Sie UpdateImageSetMetadata mit einem AWS SDKoder CLI

Die folgenden Codebeispiele zeigen, wie man es benutztUpdateImageSetMetadata.

CLI**AWS CLI**

Beispiel 1: Um ein Attribut in Bildsatz-Metadaten einzufügen oder zu aktualisieren

Im folgenden update-image-set-metadata Beispiel wird ein Attribut in Bildsatz-Metadaten eingefügt oder aktualisiert.

```
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: Um ein Attribut aus den Metadaten eines Bildsatzes zu entfernen

Im folgenden `update-image-set-metadata` Beispiel 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: Um eine Instanz aus den Metadaten eines Bildsatzes zu entfernen

Im folgenden `update-image-set-metadata` Beispiel wird eine Instanz aus den Metadaten des 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\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}}"
  }
}
```

Ausgabe:

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

Beispiel 4: Um einen Bildsatz auf eine frühere Version zurückzusetzen

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: Um einer Instanz ein privates DICOM Datenelement hinzuzufügen

Das folgende update-image-set-metadata Beispiel zeigt, wie ein privates Element zu einer angegebenen Instanz innerhalb eines Bilddatensatzes hinzugefügt wird. Der DICOM Standard erlaubt private Datenelemente für die Übertragung von Informationen, die nicht in Standarddatenelementen enthalten sein können. 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\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"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: Um ein privates DICOM Datenelement auf eine Instanz zu aktualisieren

Das folgende update-image-set-metadata Beispiel zeigt, wie der Wert eines privaten Datenelements aktualisiert wird, das zu einer Instanz innerhalb eines Bilddatensatzes gehört.

```
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 a SOPInstanceUID mit dem Force-Parameter zu aktualisieren

Das folgende update-image-set-metadata Beispiel zeigt, wie eine aktualisiert wird SOPInstanceUID, 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 unter [Aktualisieren von Bilddatensatz-Metadaten](#) im AWS HealthImaging Entwicklerhandbuch.

- API-Einheiten finden Sie [UpdateImageSetMetadata](#) in 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()
                .updateableAttributes(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: Eine Instanz entfernen.

```

        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);

```

- API-Einheiten finden Sie unter [UpdateImageSetMetadata](#) AWS SDK for Java 2.x API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 die Aktualisierung.

```
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: Eine Instanz entfernen.

```

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);
```

- API-Einheiten finden Sie unter [UpdateImageSetMetadata](#) AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

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 Objekt. `MedicalImagingWrapper`

```

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: Eine Instanz entfernen.

```
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
)
```

- API-Einheiten finden Sie unter [UpdateImageSetMetadata](#) AWS SDK für Python (Boto3) API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging 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 mit implementierten 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 Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes finden.

Die Szenarien zielen auf ein mittleres Erfahrungsniveau ab, damit Sie Serviceaktionen im Kontext besser verstehen können.

Beispiele

- [Beginnen Sie mit HealthImaging-Bilddatensätzen und Bildrahmen mithilfe eines AWS SDK](#)
- [Kennzeichnen eines HealthImaging-Datenspeichers mit einem AWS SDK](#)
- [Markieren eines HealthImaging-Bilddatensatzes mit einem AWS SDK](#)

Beginnen Sie mit HealthImaging Bilddatensätzen und Bildrahmen mithilfe eines AWS SDK

Die folgenden Codebeispiele zeigen, wie Sie DICOM Dateien importieren und Bildrahmen in heruntergeladen HealthImaging.

Die Implementierung ist als Workflow-Befehlszeilenanwendung strukturiert.

- Richten Sie Ressourcen für einen Import einDICOM.
- DICOMImportiert 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 Bildrahmen herunter, dekodieren Sie sie und überprüfen Sie sie.
- Ressourcen bereinigen.

C++

SDKfür C++

Erstellen Sie ein AWS CloudFormation Stapel 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 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 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;
}

```

Ruft Bildsätze ab, die durch den DICOM Importjob 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;
```

```
}

```

Ruft Bildrahmeninformationen für Bilddatensätze 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 outputPath: The path where the metadata will be stored as gzipped
 json.
 \param clientConfig: Aws client configuration.
 \\return bool: Function succeeded.
 */
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                    const Aws::String &imageSetID,
                                                    const Aws::String &versionID,
                                                    const Aws::String
&outputFilePath,
                                                    const
Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    if (!versionID.empty()) {
        request.SetVersionId(versionID);
    }
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
client.GetImageSetMetadata(
    request);
    if (outcome.IsSuccess()) {
        std::ofstream file(outputFilePath, std::ios::binary);
        auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
        file << metadata.rdbuf();
    }
    else {
        std::cerr << "Failed to get image set metadata: "
            << outcome.GetError().GetMessage() << std::endl;
    }
}

```

```

    return outcome.IsSuccess();
}

```

Laden Sie Bildrahmen herunter, dekodieren und verifizieren 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;
}
```

Ressourcen bereinigen.

```
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;
}
```

- APINähere Informationen finden Sie in den folgenden Themen AWS SDK for C++ APIReferenz.
 - [DeleteImageSet](#)

- [GetDICOMImport Job](#)
- [GetImageFrame](#)
- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [StartDICOMImport Job](#)

 Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

index.js - Schritte orchestrieren.

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "../deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
```

```
    outputCopiedObjects,
  } from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
```

```
    loadState,
    doCopy,
    selectDataset,
    copyDataset,
    outputCopiedObjects,
    doImport,
    startDICOMImport,
    waitForImportJobCompletion,
    outputImportJobStatus,
    getManifestFile,
    parseManifestFile,
    outputImageSetIds,
    getImageSetMetadata,
    outputImageFrameIds,
    doVerify,
    decodeAndVerifyImages,
    saveState,
  ],
  context,
),
destroy: new Scenario(
  "Clean Up Resources",
  [loadState, confirmCleanup, deleteImageSets, deleteStack],
  context,
),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js - Ressourcen bereitstellen.

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
```

```
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
```



```
    },
    {
      skipWhen: (/** @type {} */ state) => !state.deployStack,
    },
  );

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {} */ state) => {
    const stackName = state.getStackName;
    const datastoreId = state.getDatastoreId;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreId",
          ParameterValue: datastoreId,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
    });
  });
```

```

    const stack = response.Stacks?.find(
      (s) => s.StackName == state.getStackName,
    );
    if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
      throw new Error("Stack creation is still in progress");
    }
    if (stack.StackStatus === "CREATE_COMPLETE") {
      state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
        acc[output.OutputKey] = output.OutputValue;
        return acc;
      }, {});
    } else {
      throw new Error(
        `Stack creation failed with status: ${stack.StackStatus}`,
      );
    }
  });
},
{
  skipWhen: (/** @type {} */ state) => !state.deployStack,
},
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
     string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
    Datastore ID: ${stackOutputs?.DatastoreID}
    Bucket Name: ${stackOutputs?.BucketName}
    Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {} */ state) => !state.deployStack },
);

```

dataset-steps.js - DICOM Dateien kopieren.

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
```

```
(state) => {
  if (!state.doCopy) {
    process.exit(0);
  }
  return "Select a DICOM dataset to import:";
},
{
  type: "select",
  choices: datasetOptions,
},
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey}
        .split("/")
        .slice(1)
        .join("/")}`;
    });
  }
);
```

```

    const copyCommand = new CopyObjectCommand({
      Bucket: inputBucket,
      CopySource: `/${sourceBucket}/${sourceKey}`,
      Key: destinationKey,
    });

    return s3Client.send(copyCommand);
  });

  const results = await Promise.all(copyPromises);
  state.copiedObjects = results.length;
},
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`
);

```

import-steps.js - Startet den Import in den Datenspeicher.

```

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}`,
);

```

image-set-steps.js - Holen Sie sich das Bildset. IDs

```

import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
    ScenarioAction,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
 * [] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(

```

```
"getManifestFile",
async (/** @type {State} */ state) => {
  const bucket = state.stackOutputs.BucketName;
  const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
  const key = `${prefix}job-output-manifest.json`;

  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
  });

  const response = await s3Client.send(command);
  const manifestContent = await response.Body.transformToString();
  state.manifestContent = JSON.parse(manifestContent);
},
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js - Holen Sie sich den BildrahmenIDs.


```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */
```

```
/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
```

```

        await response.imageSetMetadataBlob.transformToByteArray();
        const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
        const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

        outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  /** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
  "\n",
)}\n\n`;
    }

    return output;
  },
  { slow: false },
);

```

verify-steps.js - Überprüfen Sie die Bildrahmen. Die [AWS HealthImaging Die](#) Bibliothek zur Überprüfung von 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 [seriesInstanceId, series] of Object.entries(
```

```
    metadata.Study.Series,
  )) {
    for (const [sopInstanceId, _] of Object.entries(series.Instances)) {
      console.log(
        `Verifying image set ${imageSetId} with series ${seriesInstanceId}
and sop ${sopInstanceId}`,
      );
      const child = spawn(
        "node",
        [
          verificationTool,
          datastoreId,
          imageSetId,
          seriesInstanceId,
          sopInstanceId,
        ],
        { stdio: "inherit" },
      );

      await new Promise((resolve, reject) => {
        child.on("exit", (code) => {
          if (code === 0) {
            resolve();
          } else {
            reject(
              new Error(
                `Verification tool exited with code ${code} for image set
${imageSetId}`,
              ),
            );
          }
        });
      });
    }
  }
},
);
```

clean-up-steps.js - Ressourcen zerstören.

```
import {
```

```
    CloudFormationClient,
    DeleteStackCommand,
  } from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
```

```
* @property {Object} DICOM
* @property {Series[]} Series
*/

/**
* @typedef {Object} Patient
* @property {Object} DICOM
*/

/**
* @typedef {{
*   SchemaVersion: string,
*   DatastoreID: string,
*   ImageSetID: string,
*   Patient: Patient,
*   Study: Study
* }} ImageSetMetadata
*/

/**
* @typedef {{ stackOutputs: {
*   BucketName: string,
*   DatastoreID: string,
*   RoleArn: string
* }, imageSetMetadata: ImageSetMetadata[] }} State
*/

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
```



```

        imageSetId: metadata.ImageSetID,
    });

    try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
    } catch (e) {
        if (e instanceof Error) {
            if (e.name === "ConflictException") {
                console.log(`Image set ${metadata.ImageSetID} already deleted`);
            }
        }
    }
},
{
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
},
);

export const deleteStack = new ScenarioAction(
    "deleteStack",
    async (/** @type {State} */ state) => {
        const stackName = state.getStackName;

        const command = new DeleteStackCommand({
            StackName: stackName,
        });

        await cfnClient.send(command);
        console.log(`Stack ${stackName} deletion initiated`);
    },
    {
        skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
    },
);

```

- API-Einheiten finden Sie in den folgenden Themen [AWS SDK for JavaScript API-Referenz](#).
- [DeleteImageSet](#)
- [GetDICOMImportJob](#)
- [GetImageFrame](#)

- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [StartDICOMImport Job](#)

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Erstelle ein AWS CloudFormation Stapel 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(
        "../../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml"
    ) as setup_file:
        setup_template = setup_file.read()
    print(f"\t\tCreating {stack_name}.")
    stack = self.cf_resource.create_stack(
        StackName=stack_name,
        TemplateBody=setup_template,
```

```

        Capabilities=["CAPABILITY_NAMED_IAM"],
        Parameters=[
            {
                "ParameterKey": "datastoreName",
                "ParameterValue": data_store_name,
            },
            {
                "ParameterKey": "userAccountID",
                "ParameterValue": account_id,
            },
        ],
    )
    print("\t\tWaiting for stack to deploy. This typically takes a minute or
two.")
    waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
    waiter.wait(StackName=stack.name)
    stack.load()
    print(f"\t\tStack status: {stack.stack_status}")

    outputs_dictionary = {
        output["OutputKey"]: output["OutputValue"] for output in
stack.outputs
    }
    self.input_bucket_name = outputs_dictionary["BucketName"]
    self.output_bucket_name = outputs_dictionary["BucketName"]
    self.role_arn = outputs_dictionary["RoleArn"]
    self.data_store_id = outputs_dictionary["DatastoreID"]
    return stack

```

Kopieren Sie DICOM Dateien in den Amazon S3 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 S3-Datenspeicher.

```

class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

```

```
def __init__(self, medical_imaging_client, s3_client):
    """
    :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
    :param s3_client: A Boto3 S3 client.
    """
    self.medical_imaging_client = medical_imaging_client
    self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

    def start_dicom_import_job(
        self,
        data_store_id,
        input_bucket_name,
        input_directory,
        output_bucket_name,
        output_directory,
        role_arn,
    ):
        """
        Routine which starts a HealthImaging import job.

        :param data_store_id: The HealthImaging data store ID.
        :param input_bucket_name: The name of the Amazon S3 bucket containing the
        DICOM files.
        :param input_directory: The directory in the S3 bucket containing the
        DICOM files.
        :param output_bucket_name: The name of the S3 bucket for the output.
        :param output_directory: The directory in the S3 bucket to store the
        output.
        :param role_arn: The ARN of the IAM role with permissions for the import.
        :return: The job ID of the import.
        """

        input_uri = f"s3://{input_bucket_name}/{input_directory}/"
        output_uri = f"s3://{output_bucket_name}/{output_directory}/"
        try:
            job = self.medical_imaging_client.start_dicom_import_job(
                jobName="examplejob",
```

```
        datastoreId=data_store_id,
        dataAccessRoleArn=role_arn,
        inputS3Uri=input_uri,
        outputS3Uri=output_uri,
    )
except ClientError as err:
    logger.error(
        "Couldn't start DICOM import job. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return job["jobId"]
```

Ruft Bildsätze ab, die durch den DICOM Importjob erstellt wurden.

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
        """
        Retrieves the image sets created for an import job.
```

```
:param datastore_id: The HealthImaging data store ID
:param import_job_id: The import job ID
:return: List of image set IDs
"""

import_job = self.medical_imaging_client.get_dicom_import_job(
    datastoreId=datastore_id, jobId=import_job_id
)

output_uri = import_job["jobProperties"]["outputS3Uri"]

bucket = output_uri.split("/")[2]
key = "/" .join(output_uri.split("/")[3:])

# Try to get the manifest.
retries = 3
while retries > 0:
    try:
        obj = self.s3_client.get_object(
            Bucket=bucket, Key=key + "job-output-manifest.json"
        )
        body = obj["Body"]
        break
    except ClientError as error:
        retries = retries - 1
        time.sleep(3)
try:
    data = json.load(body)
    expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
    image_sets = expression.search(data)
except json.decoder.JSONDecodeError as error:
    image_sets = import_job["jobProperties"]

return image_sets

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

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

```

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

```

Ruft Bildrahmeninformationen für Bilddatensätze ab.

```

class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")

```



```

s3_client = boto3.client("s3")
return cls(medical_imaging_client, s3_client)

def get_image_frames_for_image_set(self, datastore_id, image_set_id,
out_directory):
    """
    Get the image frames for an image set.

    :param datastore_id: The ID of the data store.
    :param image_set_id: The ID of the image set.
    :param out_directory: The directory to save the file.
    :return: The image frames.
    """
    image_frames = []
    file_name = os.path.join(out_directory,
f"{image_set_id}_metadata.json.gzip")
    file_name = file_name.replace("/", "\\")
    self.get_image_set_metadata(file_name, datastore_id, image_set_id)
    try:
        with gzip.open(file_name, "rb") as f_in:
            doc = json.load(f_in)
            instances = jmespath.search("Study.Series.*.Instances[*]", doc)
            for instance in instances:
                rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
                rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)

                image_frames_json = jmespath.search("ImageFrames[][]", instance)
                for image_frame in image_frames_json:
                    checksum_json = jmespath.search(
                        "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",

                        image_frame,
                    )
                    image_frame_info = {
                        "imageSetId": image_set_id,
                        "imageFrameId": image_frame["ID"],
                        "rescaleIntercept": rescale_intercept,
                        "rescaleSlope": rescale_slope,
                        "minPixelValue": image_frame["MinPixelValue"],
                        "maxPixelValue": image_frame["MaxPixelValue"],
                        "fullResolutionChecksum": checksum_json["Checksum"],
                    }
                    image_frames.append(image_frame_info)

```

```
        return image_frames
    except TypeError:
        return {}
    except ClientError as err:
        logger.error(
            "Couldn't get image frames for image set. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    return image_frames

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

    :param metadata_file: The file to store the JSON gzipped metadata.
    :param datastore_id: The ID of the data store.
    :param image_set_id: The ID of the image set.
    :param version_id: The version of the image set.
    """

    try:
        if version_id:
            image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                versionId=version_id,
            )
        else:
            image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
        with open(metadata_file, "wb") as f:
            for chunk in
image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                if chunk:
                    f.write(chunk)
```

```

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

```

Laden Sie Bildrahmen herunter, dekodieren und verifizieren Sie sie.

```

class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

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

        :param file_path_to_write: The path to write the image frame's HTJ2K
        encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.

```

```

    """
    try:
        image_frame = self.medical_imaging_client.get_image_frame(
            datastoreId=datastore_id,
            imageSetId=image_set_id,
            imageFrameInformation={"imageFrameId": image_frame_id},
        )
        with open(file_path_to_write, "wb") as f:
            for chunk in image_frame["imageFrameBlob"].iter_chunks():
                f.write(chunk)
    except ClientError as err:
        logger.error(
            "Couldn't get image frame. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

def download_decode_and_check_image_frames(
    self, data_store_id, image_frames, out_directory
):
    """
    Downloads image frames, decodes them, and uses the checksum to validate
    the decoded images.

    :param data_store_id: The HealthImaging data store ID.
    :param image_frames: A list of dicts containing image frame information.
    :param out_directory: A directory for the downloaded images.
    :return: True if the function succeeded; otherwise, False.
    """
    total_result = True
    for image_frame in image_frames:
        image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
        self.get_pixel_data(
            image_file_path,
            data_store_id,
            image_frame["imageSetId"],
            image_frame["imageFrameId"],
        )

        image_array = self.jph_image_to_opj_bitmap(image_file_path)
        crc32_checksum = image_frame["fullResolutionChecksum"]

```

```

        # Verify checksum.
        crc32_calculated = zlib.crc32(image_array)
        image_result = crc32_checksum == crc32_calculated
        print(
            f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
{image_result }"
        )
        total_result = total_result and image_result
    return total_result

    @staticmethod
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array

```

Ressourcen bereinigen.

```

def destroy(self, stack):
    """
    Destroys the resources managed by the CloudFormation stack, and the
    CloudFormation
    stack itself.

    :param stack: The CloudFormation stack that manages the example
    resources.
    """

    print(f"\t\tCleaning up resources and {stack.name}.")
    data_store_id = None
    for opout in stack.outputs:
        if opout["OutputKey"] == "DatastoreID":

```

```

        data_store_id = oput["OutputValue"]
    if data_store_id is not None:
        print(f"\t\tDeleting image sets in data store {data_store_id}.")
        image_sets = self.medical_imaging_wrapper.search_image_sets(
            data_store_id, {}
        )
        image_set_ids = [image_set["imageSetId"] for image_set in image_sets]

        for image_set_id in image_set_ids:
            self.medical_imaging_wrapper.delete_image_set(
                data_store_id, image_set_id
            )
            print(f"\t\tDeleted image set with id : {image_set_id}")

    print(f"\t\tDeleting {stack.name}.")
    stack.delete()
    print("\t\tWaiting for stack removal. This may take a few minutes.")
    waiter = self.cf_resource.meta.client.get_waiter("stack_delete_complete")
    waiter.wait(StackName=stack.name)
    print("\t\tStack delete complete.")

```

```

class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

    def search_image_sets(self, datastore_id, search_filter):
        """

```

```
Search for image sets.

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

def delete_image_set(self, datastore_id, image_set_id):
    """
    Delete an image set.

    :param datastore_id: The ID of the data store.
    :param image_set_id: The ID of the image set.
    """
    try:
        delete_results = self.medical_imaging_client.delete_image_set(
            imageSetId=image_set_id, datastoreId=datastore_id
        )
    except ClientError as err:
        logger.error(
            "Couldn't delete image set. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
```

```
)  
    raise
```

- APINähere Informationen finden Sie in den folgenden Themen AWS SDK für Python (Boto3) API -Referenz.
 - [DeleteImageSet](#)
 - [GetDICOMImportJob](#)
 - [GetImageFrame](#)
 - [GetImageSetMetadata](#)
 - [SearchImageSets](#)
 - [StartDICOMImportJob](#)

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Kennzeichnen eines HealthImaging Datenspeichers mit einem AWS SDK

Die folgenden Codebeispiele zeigen, wie Sie einen HealthImaging Datenspeicher taggen.

Java

SDK für Java 2.x

Um einen Datenspeicher zu taggen.

```
        final String datastoreArn = "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012";  
  
        TagResource.tagMedicalImagingResource(medicalImagingClient,  
        datastoreArn,
```



```
ImmutableMap.of("Deployment", "Development")));
```

Die Hilfsfunktion zum Markieren 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);
    }
}
```

Um Tags für einen Datenspeicher aufzulisten.

```
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 der 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;
}
```

Um die Markierung eines Datenspeichers aufzuheben.

```
final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

UntagResource.untagMedicalImagingResource(medicalImagingClient,
    datastoreArn,
        Collections.singletonList("Deployment"));
```

Die Hilfsfunktion zum Entkennzeichnen 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);
    }
}
```

- APIEinzelheiten finden Sie in den folgenden Themen unter AWS SDK for Java 2.x APIReferenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

Um einen Datenspeicher zu taggen.

```
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 Markieren 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;
};
```

Um Tags für einen Datenspeicher aufzulisten.

```
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 der 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;  
};
```

Um die Markierung eines Datenspeichers aufzuheben.

```
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 Entkennzeichnen 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;
};

```

- API-Einheiten finden Sie in den folgenden Themen unter AWS SDK for JavaScript API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Um einen Datenspeicher zu taggen.

```
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 Markieren 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
```

Um Tags für einen Datenspeicher aufzulisten.

```
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 der 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"]
```

Um die Markierung eines Datenspeichers aufzuheben.

```
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 Entkennzeichnen 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 Objekt. MedicalImagingWrapper

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

- API-Einheiten finden Sie in den folgenden Themen unter AWS SDK für Python (Boto3) API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Es gibt noch mehr dazu. GitHub Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Markieren eines HealthImaging-Bilddatensatzes mit einem AWS SDK

Die folgenden Codebeispiele zeigen, wie Sie einen HealthImaging-Bildsatz taggen.

Java

SDK für Java 2.x

Um einen Bilddatensatz zu taggen.

```
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 Markieren 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);
    }
}
```

Um Tags für einen Bilddatensatz aufzulisten.

```
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 der 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;
}
```

Um die Markierung eines Bilddatensatzes aufzuheben.

```
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 Entkennzeichnen 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);
    }
}
```

- API-Einheiten finden Sie in den folgenden Themen unter AWS SDK for Java 2.x API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

JavaScript

SDK für JavaScript (v3)

Um einen Bilddatensatz zu taggen.

```
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 Markieren 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;
};

```

Um Tags für einen Bilddatensatz aufzulisten.

```
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 der 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;
};

```

Um die Markierung eines Bilddatensatzes aufzuheben.

```
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 Entkennzeichnen 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;
};
```

- API-Einheiten finden Sie in den folgenden Themen unter AWS SDK for JavaScript API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Es gibt noch mehr dazu GitHub. Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Python

SDK für Python (Boto3)

Um einen Bilddatensatz zu taggen.

```
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 Markieren 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
```

Um Tags für einen Bilddatensatz aufzulisten.

```
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 der 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"]

```

Um die Markierung eines Bilddatensatzes aufzuheben.

```

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 Entkennzeichnen 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 Objekt. `MedicalImagingWrapper`

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

- API-Einheiten finden Sie in den folgenden Themen unter [AWS SDK für Python \(Boto3\) API-Referenz](#).
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Es gibt noch mehr dazu. [GitHub](#) Finden Sie das vollständige Beispiel und erfahren Sie, wie Sie es einrichten und ausführen in der [AWS Repository mit Codebeispielen](#).

Für eine vollständige Liste von AWS SDK-Entwicklerhandbüchern und Codebeispielen finden Sie unter [Verwendung HealthImaging mit einem AWS SDK](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden DICOMweb mit AWS HealthImaging

Sie können DICOM Objekte AWS HealthImaging mithilfe einer Repräsentation von [DICOMweb](#) APIs abrufen. Dabei handelt es sich um webbasierte Dienste, die dem DICOM Standard für medizinische Bildgebung entsprechen. Mit dieser Funktion können Sie mit Systemen zusammenarbeiten, die Binärdateien von DICOM Teil 10 verwenden, und gleichzeitig die Vorteile der [Cloud-nativen HealthImaging](#) Aktionen nutzen. Der Schwerpunkt dieses Kapitels liegt darauf, wie Sie die Implementierung HealthImaging von DICOMweb Diensten nutzen können, um Antworten zurückzugeben DICOMweb.

Wichtig

HealthImaging Importiert Jobs, verarbeitet DICOM Instanz-Binärdateien (.dcmDateien) und wandelt sie in [Bilddatensätze](#) um. Verwenden Sie HealthImaging [Cloud-native Aktionen](#) (APIs), um Bilddatensätze zu verwalten. Verwenden Sie HealthImaging die Darstellung von DICOMweb Diensten, um DICOMweb Antworten zurückzugeben.

Die in diesem Kapitel APIs aufgelisteten Systeme entsprechen dem [DICOMweb](#) Standard für webbasierte medizinische Bildgebung. Da es sich um Repräsentationen von DICOMweb Diensten handelt, werden sie nicht über AWS CLI und AWS SDKs angeboten.

Thema

- [DICOMDaten werden abgerufen von HealthImaging](#)

DICOMDaten werden abgerufen von HealthImaging

AWS HealthImaging bietet Repräsentationen von [DICOMweb WADO-RS](#) Diensten zum Abrufen einer DICOM Instanz, Instanz-Metadaten und DICOM Instanz-Frames (Pixeln) aus einem HealthImaging [Datenspeicher](#). Die DICOMweb WADO-RS Dienste bieten Flexibilität beim Abrufen von Daten, auf denen gespeichert ist, HealthImaging und sorgen für Interoperabilität mit älteren Anwendungen.

Wichtig

HealthImaging Importaufträge verarbeiten DICOM Instanz-Binärdateien (.dcmDateien) und wandeln sie in [Bilddatensätze](#) um. Verwenden Sie HealthImaging [Cloud-native Aktionen](#)

(APIs), um Bilddatensätze zu verwalten. Verwenden Sie HealthImaging die Darstellung von DICOMweb Diensten, um DICOMweb Antworten zurückzugeben.

Die in diesem Abschnitt aufgeführten Dienste entsprechen dem [DICOMweb\(WADO-RS\) - Standard](#) für webbasierte medizinische Bildgebung. Da es sich um Repräsentationen von DICOMweb Diensten handelt, werden sie nicht über AWS CLI und angeboten. AWS SDKs

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

HealthImaging Repräsentationen von -RS-Diensten DICOMweb WADO

Name	Beschreibung
GetDICOMInstance	Rufen Sie eine DICOM Instanz (.dcmDatei) aus einem HealthImaging Datenspeicher ab, indem Sie die mit einer Ressource UIDs verknüpfte Serie, Studie und Instanz angeben.
GetDICOMInstanceMetadata	Rufen Sie DICOM Instanz-Metadaten (.jsonDatei) aus einer DICOM Instanz in einem HealthImaging Datenspeicher ab, indem Sie die mit einer Ressource UIDs verknüpfte Serie, Studie und Instanz angeben.
GetDICOMInstanceFrames	Rufen Sie einzelne Bild-Frames oder Batch-Frames (multipart Anfrage) aus einer DICOM Instanz in einem HealthImaging Datenspeicher ab, indem Sie die Serien-UID, Studien-UIDUIDs, Exemplar- und Frame-Nummern angeben, die einer Ressource zugeordnet sind.

Themen

- [Eine DICOM Instanz abrufen von HealthImaging](#)
- [DICOMInstanz-Metadaten abrufen von HealthImaging](#)
- [DICOMInstanz-Frames abrufen von HealthImaging](#)

Eine DICOM Instanz abrufen von HealthImaging

Verwenden Sie die `GetDICOMInstance` Aktion, um eine DICOM Instanz (.dcmDatei) aus einem HealthImaging [Datenspeicher](#) abzurufen, indem Sie die Serie, Studie und Instanz angeben, die der Ressource UUIDs zugeordnet sind. Sie können den [Bilddatensatz](#) angeben, aus dem eine Instanzressource abgerufen werden soll, indem Sie die Bilddatensatz-ID als Abfrageparameter angeben. Darüber hinaus können Sie die Übertragungssyntax für die Komprimierung der DICOM Daten mit Unterstützung für uncompressed (ELE) oder High-Throughput JPEG 2000 (HTJ2K) wählen.

Note

Mit können Sie mit Systemen zusammenarbeiten `GetDICOMInstance`, die Binärdateien von DICOM Teil 10 verwenden, und gleichzeitig die Vorteile der Cloud-nativen Aktionen HealthImaging nutzen.

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 `studyInstanceUIDseriesInstanceUID`, und abzurufen `sopInstanceUID`. Weitere Informationen finden Sie unter [Metadaten von Bilddatensätzen abrufen](#).
3. Konstruieren Sie eine URL für die Anfrage mit den Werten für `datastoreId` `studyInstanceUIDseriesInstanceUID`, `sopInstanceUID`, und `imageSetId`. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL Pfad im folgenden Beispiel anzuzeigen. URL Das hat die Form:

```
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 (.dcmDatei) 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?  
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

Das `transfer-syntax` UID ist optional und standardmäßig auf Explicit VR Little Endian eingestellt, falls es nicht enthalten ist. Zu den unterstützten Übertragungssyntaxen gehören:

- Expliziter VR Little Endian (ELE) - 1.2.840.10008.1.2.1 (Standard)
- High-Throughput JPEG 2000 mit RPCL Optionen zur Bildkomprimierung (nur verlustfrei) - 1.2.840.10008.1.2.4.202

Weitere Informationen finden Sie unter [HTJ2K-Dekodierungsbibliotheken für AWS HealthImaging](#).

DICOMInstanz-Metadaten 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 Ressource UIDs zugeordnet sind. Sie können den [Bilddatensatz](#) angeben, aus dem die Metadaten der Instanzressource abgerufen werden sollen, indem Sie die Bilddatensatz-ID als Abfrageparameter angeben.

Note

Mit können Sie mit Systemen zusammenarbeiten `GetDICOMInstanceMetadata`, die Binärdateien von DICOM Teil 10 verwenden, und gleichzeitig die Vorteile der [Cloud-nativen HealthImaging](#) Aktionen nutzen.

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 `studyInstanceUID`, `seriesInstanceUID`, und `sopInstanceUID`. Weitere Informationen finden Sie unter [Metadaten von Bilddatensätzen abrufen](#).
3. Konstruieren Sie eine URL für die Anfrage mit den Werten für `datastoreId`, `studyInstanceUID`, `seriesInstanceUID`, `sopInstanceUID`, und `imageSetId`. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL Pfad im folgenden Beispiel anzuzeigen. URL Das hat die Form:

```
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 DICOM Instanzmetadaten (`.jsonDatei`) 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/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'
```

DICOMInstanz-Frames abrufen von HealthImaging

Verwenden Sie die `GetDICOMInstanceFrames` Aktion, um einzelne oder stapelweise Bild-Frames (`multipartAnfrage`) aus einer DICOM Instanz in einem HealthImaging [Datenspeicher](#) abzurufen, indem Sie die Serien-UID, Studien- UIDUIDs, Instanz- und Frame-Nummern angeben, die einer Ressource zugeordnet sind. Sie können den [Bilddatensatz](#) angeben, aus dem Instanzframes abgerufen werden sollen, indem Sie die Bilddatensatz-ID als Abfrageparameter angeben. Darüber hinaus können Sie die Übertragungssyntax für die Komprimierung der Image-Frame-Daten mit Unterstützung für uncompressed (ELE) oder High-Throughput JPEG 2000 (HTJ2K) wählen.

Note

Mit können Sie mit Systemen zusammenarbeiten `GetDICOMInstanceFrames`, die Binärdateien von DICOM Teil 10 verwenden, und gleichzeitig die Vorteile der Cloud-nativen Aktionen HealthImaging nutzen.

Um DICOM Instanz-Frames () zu erhalten **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 `abzurufensopInstanceUID`. 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`,

undframeList. Scrollen Sie über die Schaltfläche Kopieren, um den gesamten URL Pfad im folgenden Beispiel anzuzeigen. URLDas hat die Form:

```
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. GetDICOMInstanceFramesverwendet 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

Das transfer-syntax UID ist optional und standardmäßig auf Explicit VR Little Endian eingestellt, falls es nicht enthalten ist. Zu den unterstützten Übertragungssyntaxen gehören:

- Expliziter VR Little Endian (ELE) - 1.2.840.10008.1.2.1 (Standard)
- High-Throughput JPEG 2000 mit RPCL Optionen zur Bildkomprimierung (nur verlustfrei) - 1.2.840.10008.1.2.4.202

Weitere Informationen finden Sie unter [HTJ2K-Dekodierungsbibliotheken für AWS HealthImaging](#).

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 CloudTrail erfasst 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 Aufrufe 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 Amazon EC2 EC2-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](#).

Erstellen eines Trails

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 aktuelle Ereignisse in Ihrem anzeigen, suchen und herunterladen AWS-Konto. Weitere Informationen finden Sie unter [Ereignisse mit CloudTrail Ereignisverlauf anzeigen](#).

Note

Um den CloudTrail Ereignisverlauf für AWS HealthImaging in der anzuzeigen AWS Management Console, rufen Sie das Menü Suchattribute auf, wählen Sie Ereignisquelle aus und wählen `Siemedia1-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](#)
- [CloudTrail unterstützte Dienste 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, einschließlich 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 APIs vorbehalten, die hohe Transaktionen pro Sekunde (tps) aufweisen. Sie können also aus Kostengründen die CloudTrail Protokollierung deaktivieren.

Mit HealthImaging Ausnahme von gelten alle in der [HealthImaging AWS-API-Referenz](#) aufgeführten API-Aktionen als Verwaltungsereignisse `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 .

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 userIdentityElement](#).

Logeinträge verstehen

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": "XXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
    "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
    "accountId": "123456789012",
    "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "XXXXXXXXXXXXXXXXXXXX",
        "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
        "accountId": "123456789012",
        "userName": "TestAccessRole"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-10-28T15:52:42Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2022-10-28T16:02:30Z",
  "eventSource": "medical-imaging.amazonaws.com",
  "eventName": "GetDICOMImportJob",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
```

```
"userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-  
Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync  
http/Apache cfg/retry-mode/standard",  
  "requestParameters": {  
    "jobId": "5d08d05d6aab2a27922d6260926077d4",  
    "datastoreId": "12345678901234567890123456789012"  
  },  
  "responseElements": null,  
  "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",  
  "eventID": "26307f73-07f4-4276-b379-d362aa303b22",  
  "readOnly": true,  
  "eventType": "AwsApiCall",  
  "managementEvent": true,  
  "recipientAccountId": "824333766656",  
  "eventCategory": "Management"  
}
```

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](#).

Note

Metriken werden für alle HealthImaging APIs gemeldet.

In den folgenden Tabellen sind die Metriken und Dimensionen für aufgeführt HealthImaging. Jede Zahl wird als Häufigkeitszahl für einen vom Benutzer angegebenen Datenbereich dargestellt.

Metriken

Metriken	Beschreibung
Anzahl der Anrufe	<p>Die Anzahl der Aufrufe von APIs. Dies kann entweder für das Konto oder einen bestimmten Datenspeicher gemeldet werden.</p> <p>Einheiten: Anzahl</p> <p>Gültige Statistiken: Summe, Anzahl</p> <p>Dimensionen: Vorgang, Datenspeicher-ID, Datenspeichertyp</p>

Sie können Metriken für HealthImaging mit der AWS Management Console 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 Metriken anzeigen

Um Metriken anzuzeigen (CloudWatch Konsole)

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die [CloudWatch Konsole](#).
2. Wählen Sie Metrics, All Metrics und anschließend AWS/Medical Imaging aus.
3. Wählen Sie die Dimension, den Namen einer Metrik und schließlich Add to graph (Dem Diagramm hinzufügen) aus.
4. Wählen Sie einen Wert für den Datumsbereich aus. Die Anzahl der Kennzahlen für den ausgewählten Zeitraum wird im Diagramm 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.

Themen

- [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	
Datenspeicher erstellen	CREATING

HealthImaging Ereignistyp	Status
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
Aktualisierung des Bildsatzes	UPDATING

HealthImaging Ereignistyp	Status
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 [ImageSetWorkflowStatus](#) in 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	Beschreibung
version	Zeichenfolge	Die Version des EventBridge Ereignisschemas.
id	Zeichenfolge	Die für jedes Ereignis generierte UUID der Version 4.
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	Beschreibung
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	Beschreibung
<code>version</code>	Zeichenfolge	Die Version des EventBridge Ereignisschemas.
<code>id</code>	Zeichenfolge	Die für jedes Ereignis generierte UUID der Version 4.
<code>detail-type</code>	Zeichenfolge	Der Ereignistyp, der gesendet wird.
<code>source</code>	Zeichenfolge	Identifiziert den Service, aus dem das Ereignis stammt.
<code>account</code>	Zeichenfolge	Die 12-stellige AWS-Konto-ID des Datenspeicher-Besitzers.
<code>time</code>	Zeichenfolge	Der Zeitpunkt, zu dem das Ereignis eingetreten ist.
<code>region</code>	Zeichenfolge	Identifiziert die AWS Region des Datenspeichers.
<code>resources</code>	Array (Zeichenfolge)	Ein JSON-Array, das den ARN des Datenspeichers enthält.
<code>detail</code>	object	Ein JSON-Objekt, das Informationen zum Ereignis enthält.
<code>detail.imagingVersion</code>	Zeichenfolge	Die Versions-ID, die Änderungen HealthImaging am Ereignisdetailschema verfolgt.

Name	Typ	Beschreibung
<code>detail.datastoreId</code>	Zeichenfolge	Der Datenspeicher, der das Statusänderungsereignis generiert hat.
<code>detail.jobId</code>	Zeichenfolge	Die mit dem Statusänderungsereignis verknüpfte Importjob-ID.
<code>detail.jobName</code>	Zeichenfolge	Der Name des Importauftrags.
<code>detail.jobStatus</code>	Zeichenfolge	Der aktuelle Jobstatus.
<code>detail.inputS3Uri</code>	Zeichenfolge	Der Eingabeprefixpfad für den S3-Bucket, der die zu importierenden DICOM-Datensätze enthält.
<code>detail.outputS3Uri</code>	Zeichenfolge	Das Ausgabepräfix des S3-Buckets, in den die Ergebnisse 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",
  "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",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "COPYING"
  }
}
```

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",
  "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",
    "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",
  "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",
    "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",
  "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",
    "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",
    "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",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "DELETED",
    "imageSetWorkflowStatus": "DELETED"
  }
}
```

Bilderdaten-Ereignisse — Metadaten-Beschreibungen

Name	Typ	Beschreibung
<code>version</code>	Zeichenfolge	Die Version des EventBridge Ereignisschemas.
<code>id</code>	Zeichenfolge	Die für jedes Ereignis generierte UUID der Version 4.
<code>detail-type</code>	Zeichenfolge	Der Ereignistyp, der gesendet wird.
<code>source</code>	Zeichenfolge	Identifiziert den Service, aus dem das Ereignis stammt.
<code>account</code>	Zeichenfolge	Die 12-stellige AWS-Konto-ID des Datenspeicher-Besitzers.
<code>time</code>	Zeichenfolge	Der Zeitpunkt, zu dem das Ereignis eingetreten ist.
<code>region</code>	Zeichenfolge	Identifiziert die AWS Region des Datenspeichers.
<code>resources</code>	Array (Zeichenfolge)	Ein JSON-Array, das den ARN des Bildsatzes enthält.
<code>detail</code>	object	Ein JSON-Objekt, das Informationen zum Ereignis enthält.
<code>detail.imagingVersion</code>	Zeichenfolge	Die Versions-ID, die Änderungen HealthImaging am Ereignisdetailschema verfolgt.

Name	Typ	Beschreibung
<code>detail.datastoreId</code>	Zeichenfolge	Die Datenspeicher-ID, die das Statusänderungsereignis generiert hat.
<code>detail.imagesetId</code>	Zeichenfolge	Die Bilddatensatz-ID, die dem Statusänderungsereignis zugeordnet ist.
<code>detail.imageSetState</code>	Zeichenfolge	Der aktuelle Status des Bilddatensatzes.
<code>detail.imageSetWorkflowStatus</code>	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 selbst 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 gelten AWS HealthImaging, finden Sie unter [AWS Services im Umfang nach Compliance-Programmen AWS](#) .
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, 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.

Themen

- [Datenschutz in AWS HealthImaging](#)
- [Identitäts- und Zugriffsverwaltung 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 Tool AWS Modell der [gemeinsamen Verantwortung Modell](#) der der gemeinsamen Verantwortung gilt für den Datenschutz in AWS HealthImaging. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre Inhalte zu behalten, die auf dieser Infrastruktur gehostet werden. Sie sind auch verantwortlich für die Sicherheitskonfiguration und die Verwaltungsaufgaben für AWS-Services die Sie verwenden. Weitere Informationen zum Datenschutz finden Sie in der [Datenschutzerklärung FAQ](#). Informationen zum Datenschutz in Europa finden Sie auf der [AWS Modell der geteilten Verantwortung und GDPR](#) Blogbeitrag auf der AWS Blog zum Thema Sicherheit.

Aus Datenschutzgründen empfehlen wir Ihnen, AWS-Konto Anmeldeinformationen und richten Sie einzelne Benutzer ein mit AWS IAM Identity Center or AWS Identity and Access Management (IAM). 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).
- Verwenden Sie SSL/TLS, um mit zu kommunizieren AWS Ressourcen schützen. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Einrichtung API und Protokollierung von Benutzeraktivitäten mit AWS CloudTrail. Für Informationen zur Verwendung von CloudTrail Spuren zum Erfassen AWS Aktivitäten finden Sie unter [Arbeiten mit CloudTrail Pfaden](#) im AWS CloudTrail Benutzeranleitung.
- 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 FIPS 140-3 validierte kryptografische Module benötigen AWS über eine Befehlszeilenschnittstelle oder einen API, verwenden Sie einen Endpunkt. FIPS Weitere Informationen zu den verfügbaren FIPS Endpunkten 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. Dazu gehört auch, wenn Sie mit oder anderen arbeiten HealthImaging AWS-Services mit der Konsole API, AWS CLI, oder AWS SDKs. Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie einem externen Server eine

URL zur Verfügung stellen, empfehlen wir dringend, dass Sie keine Anmeldeinformationen angeben, URL um Ihre Anfrage an diesen Server zu validieren.

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 Fällen verfügbar AWS service
- Flexible Schlüsselverwaltungsoptionen, einschließlich AWS Key Management Service, mit denen Sie wählen können, ob Sie AWS um die Verschlüsselungsschlüssel zu verwalten oder die vollständige Kontrolle über Ihre eigenen Schlüssel zu behalten.
- AWS im Besitz AWS KMS Verschlüsselungsschlüssel
- Verschlüsselte Nachrichtenwarteschlangen für die Übertragung sensibler Daten mit serverseitiger Verschlüsselung (SSE) für Amazon SQS

Darüber hinaus AWS bietet Ihnen APIs die Möglichkeit, Verschlüsselung und Datenschutz in alle Dienste zu integrieren, die Sie entwickeln oder bereitstellen AWS Umgebung.

Verschlüsselung im Ruhezustand

HealthImaging bietet standardmäßig Verschlüsselung, um vertrauliche Kundendaten im Speicher zu schützen, indem ein eigener Dienst verwendet wird AWS KMS Schlüssel.

Verschlüsselung während der Übertragung

HealthImaging verwendet TLS 1.2, um Daten bei der Übertragung über den öffentlichen Endpunkt und über Backend-Dienste zu verschlüsseln.

Schlüsselverwaltung

AWS KMS Schlüssel (KMSSchlüssel) 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 Daten mit privaten Gesundheitsinformationen (PHI) im Ruhezustand automatisch zu verschlüsseln. AWS KMS Eigene Schlüssel werden nicht in Ihrem Konto gespeichert. Sie sind Teil einer Sammlung von KMS Schlüsseln AWS besitzt und verwaltet zur Verwendung in mehreren AWS Konten. AWS Dienste können verwenden AWS eigene KMS Schlüssel zum Schutz Ihrer Daten. Sie können es nicht ansehen, verwalten, verwenden AWS Sie besitzen KMS Schlüssel oder überprüfen deren Verwendung. Sie müssen jedoch keine Arbeit verrichten oder Programme ändern, um die Schlüssel zu schützen, mit denen Ihre Daten verschlüsselt werden.

Ihnen wird keine monatliche Gebühr oder Nutzungsgebühr berechnet, wenn Sie AWS eigene KMS Schlüssel, und sie werden nicht angerechnet AWS KMS Kontingente für Ihr Konto. Weitere Informationen finden Sie unter [AWSEigene Schlüssel](#) im AWS Key Management Service Leitfadens für Entwickler.

Vom Kunden verwaltete KMS Schlüssel

HealthImaging unterstützt die Verwendung eines symmetrischen, vom Kunden verwalteten KMS Schlüssels, den Sie selbst erstellen, besitzen und verwalten, um eine zweite Verschlüsselungsebene zu den vorhandenen hinzuzufügen AWS eigene Verschlüsselung. Da Sie die volle Kontrolle über diese Verschlüsselungsebene haben, können Sie beispielsweise folgende Aufgaben ausführen:

- Festlegung und Aufrechterhaltung 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 zu verfolgen, die HealthImaging gesendet werden an AWS KMS in Ihrem Namen. Zusätzliche AWS KMS Gebühren fallen an. Weitere

Informationen finden Sie unter [Vom Kunden verwaltete Schlüssel](#) in der AWS Key Management Service Leitfadens für Entwickler.

Einen vom Kunden verwalteten Schlüssel erstellen

Sie können einen symmetrischen, vom Kunden verwalteten Schlüssel erstellen, indem Sie den AWS Management Console oder das AWS KMS APIs. Weitere Informationen finden Sie unter [Erstellen symmetrischer KMS Verschlüsselungsschlüssel](#) 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 vom Kunden verwalteten Schlüssel erstellen, können Sie eine Schlüsselrichtlinie angeben. Weitere Informationen finden Sie unter [Verwaltung des Zugriffs auf vom Kunden verwaltete Schlüssel](#) im AWS Key Management Service Leitfadens für Entwickler.

Um Ihren vom Kunden verwalteten Schlüssel mit Ihren HealthImaging Ressourcen verwenden zu können, müssen [kms: CreateGrant](#) -Operationen in der Schlüsselrichtlinie zugelassen 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 Rechte](#) HealthImaging erhält. Weitere Informationen finden Sie unter [Grants in AWS KMS](#) in der AWS Key Management Service Leitfadens für Entwickler.

Um Ihren vom Kunden verwalteten KMS Schlüssel mit Ihren HealthImaging Ressourcen verwenden zu können, müssen die folgenden API Vorgänge in der Schlüsselrichtlinie zulässig 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*"
  ],
  "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 AWS KMS Wenn die Verschlüsselung mithilfe eines vom Kunden verwalteten KMS Schlüssels aktiviert ist, sind sowohl für die Schlüsselrichtlinie als auch für die IAM Richtlinie für den Benutzer oder die Rolle, die den HealthImaging Datenspeicher erstellt, Berechtigungen erforderlich.

Weitere Informationen zu wichtigen Richtlinien finden Sie unter [Aktivieren von IAM Richtlinien](#) im AWS Key Management Service Leitfadens für Entwickler.

Der IAM Benutzer, IAM die Rolle oder AWS Das Konto, das Ihre Repositories erstellt `kms:CreateGrant`, muss über die Berechtigungen für `kms:GenerateDataKey` `kms:RetireGrant`, `kms:Decrypt` `kms:ReEncrypt*`, und sowie über die erforderlichen Berechtigungen für AWS HealthImaging verfügen.

Wie HealthImaging verwendet man Zuschüsse in AWS KMS

HealthImaging erfordert einen [Zuschuss](#), um Ihren vom Kunden verwalteten KMS Schlüssel verwenden zu können. Wenn Sie einen Datenspeicher erstellen, der mit einem vom Kunden verwalteten KMS Schlüssel verschlüsselt ist, HealthImaging erstellt er 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 KMS Schlüssel in einem Kundenkonto zu gewähren.

Die Zuschüsse, die HealthImaging in Ihrem Namen gewährt werden, sollten nicht zurückgezogen oder zurückgezogen werden. Wenn Sie den Zuschuss widerrufen oder zurückziehen, der die HealthImaging Erlaubnis zur Nutzung des AWS KMS Schlüssel in Ihrem Konto HealthImaging können nicht auf diese Daten zugreifen, neue Imaging-Ressourcen, die in den Datenspeicher übertragen werden, verschlüsseln oder sie entschlüsseln, wenn sie abgerufen werden. Wenn Sie einen Zuschuss für widerrufen oder auslaufen lassen 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 HealthImaging gesendet werden an AWS KMS in Ihrem Namen, wenn Sie einen vom Kunden verwalteten KMS Schlüssel verwenden. Die Protokolleinträge im CloudTrail Protokoll werden `medical-imaging.amazonaws.com` in dem `userAgent` Feld angezeigt, um Anfragen von deutlich zu unterscheiden HealthImaging.

Die folgenden Beispiele sind CloudTrail Ereignisse für `CreateGrant`,, und `GenerateDataKeyDecrypt`, `DescribeKey` die überwacht werden sollen AWS KMS Operationen, die von 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 HealthImaging ermöglichen und mit diesem KMS Schlüssel alle gespeicherten Kundendaten verschlüsseln können.

Benutzer müssen keine eigenen Zuschüsse erstellen. HealthImaging erstellt in Ihrem Namen einen Zuschuss, indem Sie eine `CreateGrant` Anfrage senden an AWS KMS. Zuschüsse in AWS KMS werden verwendet, um HealthImaging Zugang zu einem zu gewähren AWS KMS geben Sie ein Kundenkonto ein.

```

{
  "Grants": [
    {
      "Operations": [
        "Decrypt",
        "Encrypt",
        "GenerateDataKey",
        "GenerateDataKeyWithoutPlaintext",
        "DescribeKey"
      ],
      "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
      "Name": "0a74e6ad2aa84b74a22fcd3efac1eaa8",
      "RetiringPrincipal": "AWS Internal",
      "GranteePrincipal": "AWS Internal",
      "GrantId":
"0da169eb18ffd3da8c0eebc9e74b3839573eb87e1e0dce893bb544a34e8fbaaf",
      "IssuingAccount": "AWS Internal",
      "CreationDate": 1685050229.0,
      "Constraints": {
        "EncryptionContextSubset": {
          "kms-arn": "arn:aws:kms:us-
west-2:824333766656:key/2fe3c119-792d-4b99-822f-b5841e1181d1"
        }
      }
    },
    {
      "Operations": [
        "GenerateDataKey",
        "CreateGrant",
        "RetireGrant",
        "DescribeKey"
      ],
      "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
      "Name": "2023-05-25T21:30:17",
      "RetiringPrincipal": "AWS Internal",
      "GranteePrincipal": "AWS Internal",
      "GrantId":
"8229757abbb2019555ba64d200278cedac08e5a7147426536fcd1f4270040a31",
      "IssuingAccount": "AWS Internal",
      "CreationDate": 1685050217.0,
    }
  ]
}

```

```
]
}
```

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 AWS KMS gehört dem Kunden AWS KMS Der Schlüssel befindet sich in einem verwendbaren Zustand und soll dem Benutzer bei der Fehlerbehebung 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 bieten weitere Informationen zur Verschlüsselung von Daten im Ruhezustand und befinden sich im [AWS Key Management Service Leitfaden für Entwickler](#).

- [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. Verkehr zwischen und HealthImaging AWS Key Management Service verwendet HTTPS standardmäßig.

- 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 Verkehr zwischen Amazon S3-Buckets HealthImaging und Amazon S3-Buckets verschlüsselt Transport Layer Security (TLS) Objekte, die zwischen HealthImaging Amazon S3 übertragen werden, HealthImaging und zwischen Kundenanwendungen, die darauf zugreifen, sollten Sie nur verschlüsselte Verbindungen über HTTPS (TLS) zulassen, indem Sie die [aws:SecureTransport condition](#) Amazon S3 S3-Bucket-Richtlinien verwenden. IAM 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 Verkehr zwischen Amazon S3 HealthImaging und Amazon S3 wird über die AWS Netzwerk und ist verschlüsselt mit TLS.

Identitäts- und Zugriffsverwaltung für AWS HealthImaging

AWS Identity and Access Management (IAM) ist ein AWS-Service das hilft einem Administrator, den Zugriff auf sicher zu kontrollieren AWS Ressourcen schützen. IAM Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um HealthImaging Ressourcen zu verwenden. IAM ist ein AWS-Service das Sie ohne zusätzliche Kosten nutzen können.

Themen

- [Zielgruppe](#)
- [Authentifizierung mit Identitäten](#)
- [Verwalten des Zugriffs mit Richtlinien](#)
- [Wie AWS HealthImaging funktioniert mit IAM](#)

- [Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging](#)
- [AWSverwaltete Richtlinien für AWS HealthImaging](#)
- [Fehlerbehebung für AWS HealthImaging -Identität und -Zugriff](#)

Zielgruppe

Wie benutzt du AWS Identity and Access Management (IAM) unterscheidet sich je nach der Arbeit, in der Sie arbeiten HealthImaging.

Dienstbenutzer — Wenn Sie den HealthImaging Dienst für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr HealthImaging Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anzufordern müssen. Wenn Sie in nicht auf eine Funktion zugreifen können HealthImaging, finden Sie weitere Informationen unter [Fehlerbehebung für AWS HealthImaging -Identität und -Zugriff](#).

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für die HealthImaging Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf HealthImaging. Es ist Ihre Aufgabe, zu bestimmen, auf welche HealthImaging Funktionen und Ressourcen Ihre Servicebenutzer zugreifen sollen. Anschließend müssen Sie Anfragen an Ihren IAM Administrator senden, um die Berechtigungen Ihrer Servicebenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die grundlegenden Konzepte von zu verstehenIAM. Weitere Informationen darüber, wie Ihr Unternehmen IAM mit verwenden kann HealthImaging, finden Sie unter [Wie AWS HealthImaging funktioniert mit IAM](#).

IAMAdministrator — Wenn Sie ein IAM Administrator sind, möchten Sie vielleicht mehr darüber erfahren, wie Sie Richtlinien schreiben können, um den Zugriff darauf zu verwalten HealthImaging. Beispiele für HealthImaging identitätsbasierte Richtlinien, die Sie in verwenden könnenIAM, finden Sie unter. [Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging](#)

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich anmelden AWS mit Ihren Identitätsdaten. Sie müssen authentifiziert (angemeldet) sein AWS) als Root-Benutzer des AWS-Kontos, als IAM Benutzer oder indem Sie eine IAM Rolle übernehmen.

Sie können sich anmelden bei AWS als föderierte Identität mithilfe von Anmeldeinformationen, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAM Identity Center) - Nutzer, die Single-Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als föderierte Identität anmelden, hat Ihr Administrator zuvor einen Identitätsverbund mithilfe von Rollen eingerichtet. IAM Wenn Sie darauf zugreifen AWS Wenn Sie den Verbund verwenden, übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich bei der anmelden AWS Management Console oder das AWS Zugangsportal. Weitere Informationen zur Anmeldung bei AWS, siehe [So melden Sie sich bei AWS-Konto](#) in der AWS-Anmeldung Benutzerleitfaden.

Wenn Sie darauf zugreifen AWS programmatisch AWS stellt ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (CLI) bereit, um Ihre Anfragen mithilfe Ihrer Anmeldeinformationen kryptografisch zu signieren. Wenn Sie es nicht verwenden AWS Tools, Sie müssen Anfragen selbst unterschreiben. Weitere Informationen zur Verwendung der empfohlenen Methode, um Anfragen selbst zu signieren, finden Sie unter [Signieren AWS APIAnfragen](#) im IAMBenutzerhandbuch.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. Zum Beispiel AWS empfiehlt, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter [Multi-Faktor-Authentifizierung](#) im AWS IAM Identity Center Benutzerhandbuch und [Verwendung der Multi-Faktor-Authentifizierung \(\) MFA in AWS](#) im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie eine erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle hat AWS-Services und Ressourcen im Konto. Diese Identität wird als AWS-Konto Root-Benutzer. Der Zugriff erfolgt, indem Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie im Benutzerhandbuch unter [Aufgaben, für die Root-Benutzeranmeldedaten erforderlich](#) sind. IAM

Verbundidentität

Es hat sich bewährt, menschlichen Benutzern, einschließlich Benutzern, die Administratorzugriff benötigen, vorzuschreiben, für den Zugriff den Verbund mit einem Identitätsanbieter zu verwenden AWS-Services mithilfe temporärer Anmeldeinformationen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensbenutzerverzeichnis, ein Web-Identitätsanbieter, der AWS Directory Service, das Identity Center-Verzeichnis oder ein beliebiger Benutzer, der zugreift AWS-Services mithilfe von Anmeldeinformationen, die über eine Identitätsquelle bereitgestellt wurden. Wenn föderierte Identitäten darauf zugreifen AWS-Konten, übernehmen sie Rollen, und die Rollen stellen temporäre Anmeldeinformationen bereit.

Für eine zentralisierte Zugriffsverwaltung empfehlen wir die Verwendung AWS IAM Identity Center. Sie können Benutzer und Gruppen in IAM Identity Center erstellen, oder Sie können eine Verbindung zu einer Gruppe von Benutzern und Gruppen in Ihrer eigenen Identitätsquelle herstellen und diese synchronisieren, um sie in all Ihren AWS-Konten und Anwendungen. Informationen zu IAM Identity Center finden Sie unter [Was ist IAM Identity Center?](#) in der AWS IAM Identity Center Benutzerleitfaden.

IAM-Benutzer und -Gruppen

Ein [IAMBenutzer](#) ist eine Identität innerhalb Ihres AWS-Konto das über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wir empfehlen, sich nach Möglichkeit auf temporäre Anmeldeinformationen zu verlassen, anstatt IAM Benutzer mit langfristigen Anmeldeinformationen wie Passwörtern und Zugriffsschlüsseln zu erstellen. Wenn Sie jedoch spezielle Anwendungsfälle haben, für die langfristige Anmeldeinformationen von IAM Benutzern erforderlich sind, empfehlen wir, die Zugriffsschlüssel abwechselnd zu verwenden. Weitere Informationen finden Sie im Benutzerhandbuch unter [Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, für die IAM langfristige Anmeldeinformationen erforderlich sind](#).

Eine [IAMGruppe](#) ist eine Identität, die eine Sammlung von IAM Benutzern spezifiziert. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise eine Gruppe benennen IAMAdminsund dieser Gruppe Berechtigungen zur Verwaltung von IAM Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen

bereit. Weitere Informationen finden Sie unter [Wann sollte ein IAM Benutzer \(statt einer Rolle\) erstellt werden?](#) im IAMBenutzerhandbuch.

IAMRollen

Eine [IAMRolle](#) ist eine Identität in deinem AWS-Konto das hat spezifische Berechtigungen. Es ähnelt einem IAM Benutzer, ist jedoch keiner bestimmten Person zugeordnet. Sie können vorübergehend eine IAM Rolle in der übernehmen AWS Management Console indem Sie die [Rollen wechseln](#). Sie können eine Rolle übernehmen, indem Sie einen anrufen AWS CLI or AWS APIOperation oder mithilfe eines benutzerdefiniertenURL. Weitere Informationen zu Methoden zur Verwendung von Rollen finden Sie [unter Verwenden von IAM Rollen](#) im IAMBenutzerhandbuch.

IAMRollen mit temporären Anmeldeinformationen sind in den folgenden Situationen nützlich:

- **Verbundbenutzerzugriff** – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie im IAMBenutzerhandbuch unter [Erstellen einer Rolle für einen externen Identitätsanbieter](#). Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Um zu kontrollieren, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in. IAM Informationen zu Berechtigungssätzen finden Sie unter [Berechtigungssätze](#) im AWS IAM Identity Center Benutzerleitfaden.
- **Temporäre IAM Benutzerberechtigungen** — Ein IAM Benutzer oder eine Rolle kann eine IAM Rolle übernehmen, um vorübergehend verschiedene Berechtigungen für eine bestimmte Aufgabe zu übernehmen.
- **Kontoübergreifender Zugriff** — Sie können eine IAM Rolle verwenden, um einer Person (einem vertrauenswürdigen Principal) in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Allerdings mit einigen AWS-Services, Sie können eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zum Unterschied zwischen Rollen und ressourcenbasierten Richtlinien für den kontenübergreifenden Zugriff finden Sie [IAMim Benutzerhandbuch unter Kontoübergreifender Ressourcenzugriff](#). IAM
- **Serviceübergreifender Zugriff** — Einige AWS-Services Funktionen in anderen verwenden AWS-Services. Wenn Sie beispielsweise einen Service aufrufen, ist es üblich, dass dieser Service Anwendungen in Amazon ausführt EC2 oder Objekte in Amazon S3 speichert. Ein Dienst kann

dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.

- Zugriffssitzungen weiterleiten (FAS) — Wenn Sie einen IAM Benutzer oder eine Rolle verwenden, um Aktionen auszuführen in AWS, Sie gelten als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FAS verwendet die Rechte des Prinzipals, der einen aufruft AWS-Service, kombiniert mit der Anfrage AWS-Service um Anfragen an nachgelagerte Dienste zu stellen. FASANfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, die Interaktionen mit anderen erfordert AWS-Services oder zu vervollständigende Ressourcen. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS Anfragen finden Sie unter [Zugriffssitzungen weiterleiten](#).
- Servicerolle — Eine Servicerolle ist eine [IAMRolle](#), die ein Dienst übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM Administrator kann eine Servicerolle von innen heraus erstellen, ändern und löschenIAM. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen an ein AWS-Service](#) im IAM-Benutzerhandbuch.
- Dienstverknüpfte Rolle — Eine dienstverknüpfte Rolle ist eine Art von Servicerolle, die mit einem verknüpft ist AWS-Service. Der Dienst kann die Rolle übernehmen, eine Aktion in Ihrem Namen durchzuführen. Mit Diensten verknüpfte Rollen erscheinen in Ihrem AWS-Konto und gehören dem Dienst. Ein IAM Administrator kann die Berechtigungen für dienstbezogene Rollen anzeigen, aber nicht bearbeiten.
- Anwendungen, die auf Amazon laufen EC2 — Sie können eine IAM Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2 Instance ausgeführt werden und AWS CLI or AWS APIAnfragen. Dies ist dem Speichern von Zugriffsschlüsseln innerhalb der EC2 Instanz vorzuziehen. Um eine zuzuweisen AWS Sie erstellen ein EC2 Instanzprofil, das an die Instanz angehängt ist. Sie müssen einer Instanz eine Rolle zuweisen und sie allen ihren Anwendungen zur Verfügung stellen. Ein Instanzprofil enthält die Rolle und ermöglicht Programmen, die auf der EC2 Instanz ausgeführt werden, temporäre Anmeldeinformationen abzurufen. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter [Verwenden einer IAM Rolle zur Erteilung von Berechtigungen für Anwendungen, die auf EC2 Amazon-Instances ausgeführt](#) werden.

Informationen darüber, ob Sie IAM Rollen oder IAM Benutzer verwenden sollten, finden [Sie im Benutzerhandbuch unter Wann sollte eine IAM Rolle \(anstelle eines IAM Benutzers\) erstellt](#) werden.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff in AWS indem Sie Richtlinien erstellen und diese anhängen AWS Identitäten oder Ressourcen. Eine Richtlinie ist ein Objekt in AWS das, wenn es mit einer Identität oder Ressource verknüpft ist, ihre Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Principal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anfrage stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien sind gespeichert in AWS als JSON Dokumente. Weitere Informationen zur Struktur und zum Inhalt von JSON Richtliniendokumenten finden Sie im IAMBenutzerhandbuch unter [Überblick über JSON Richtlinien](#).

Administratoren können Folgendes verwenden AWS JSONRichtlinien, um festzulegen, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Um Benutzern die Erlaubnis zu erteilen, Aktionen mit den Ressourcen durchzuführen, die sie benötigen, kann ein IAM Administrator IAM Richtlinien erstellen. Der Administrator kann dann die IAM Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen übernehmen.

IAMRichtlinien definieren Berechtigungen für eine Aktion, unabhängig von der Methode, mit der Sie den Vorgang ausführen. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der abrufen AWS Management Console, der AWS CLI, oder der AWS API.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind Dokumente mit JSON Berechtigungsrichtlinien, die Sie an eine Identität anhängen können, z. B. an einen IAM Benutzer, eine Benutzergruppe oder eine Rolle. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen einer identitätsbasierten Richtlinie finden Sie unter [IAMRichtlinien erstellen im Benutzerhandbuch](#). IAM

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Benutzern, Gruppen und Rollen in Ihrem AWS-Konto. Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie

Sie zwischen einer verwalteten Richtlinie oder einer Inline-Richtlinie wählen können, finden Sie im IAMBenutzerhandbuch unter [Auswahl zwischen verwalteten Richtlinien und Inline-Richtlinien](#).

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON Richtliniendokumente, die Sie an eine Ressource anhängen. Beispiele für ressourcenbasierte Richtlinien sind IAM Rollenvertrauensrichtlinien und Amazon S3 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 AWS-Services.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können nicht verwenden AWS verwaltete Richtlinien aus IAM einer ressourcenbasierten Richtlinie.

Zugriffskontrolllisten (ACLs)

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.

Amazon S3 S3, AWS WAF, und Amazon VPC sind Beispiele für Dienste, die unterstützen ACLs. Weitere Informationen finden Sie unter [Übersicht über ACLs die Zugriffskontrollliste \(ACL\)](#) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- **Berechtigungsgrenzen** — Eine Berechtigungsgrenze ist eine erweiterte Funktion, mit der Sie die maximalen Berechtigungen festlegen, die eine identitätsbasierte Richtlinie einer IAM Entität (IAMBenutzer oder Rolle) gewähren kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien

setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen zu Berechtigungsgrenzen finden Sie im IAMBenutzerhandbuch unter [Berechtigungsgrenzen für IAM Entitäten](#).

- Dienststeuerungsrichtlinien (SCPs) — SCPs sind JSON Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in festlegen AWS Organizations. AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer AWS-Konten den Ihr Unternehmen besitzt. Wenn Sie alle Funktionen in einer Organisation aktivieren, können Sie Richtlinien zur Servicesteuerung (SCPs) auf einige oder alle Ihre Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Root-Benutzer des AWS-Kontos. Weitere Informationen zu Organizations und finden Sie SCPs unter [Richtlinien zur Servicesteuerung](#) in der AWS Organizations Benutzerleitfaden.
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter [Sitzungsrichtlinien](#).

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Um zu erfahren, wie AWS bestimmt, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, siehe [Bewertungslogik für Richtlinien](#) im IAMBenutzerhandbuch.

Wie AWS HealthImaging funktioniert mit IAM

Informieren Sie sich vor der Verwendung IAM zur Verwaltung des Zugriffs auf HealthImaging, welche IAM Funktionen zur Verwendung verfügbar sind HealthImaging.

IAMFunktionen, die Sie zusammen verwenden können AWS HealthImaging

IAMMerkmal	HealthImaging Unterstützung
Identitätsbasierte Richtlinien	Ja

IAMMerkmal	HealthImaging Unterstützung
Ressourcenbasierte Richtlinien	Nein
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Richtlinienbedingungsschlüssel (servicespezifisch)	Ja
ACLs	Nein
ABAC(Tags in Richtlinien)	Teilweise
Temporäre Anmeldeinformationen	Ja
Hauptberechtigungen	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Nein

Um einen allgemeinen Überblick darüber zu erhalten, wie HealthImaging und welche AWS Dienste funktionieren mit den meisten IAM Funktionen, siehe [AWS Dienste, mit denen IAM](#) im IAMBenutzerhandbuch gearbeitet werden kann.

Identitätsbasierte Richtlinien für HealthImaging

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind Dokumente mit JSON Berechtigungsrichtlinien, die Sie an eine Identität anhängen können, z. B. an einen IAM Benutzer, eine Benutzergruppe oder eine Rolle. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen einer identitätsbasierten Richtlinie finden Sie unter [IAMRichtlinien erstellen im Benutzerhandbuch](#). IAM

Mit IAM identitätsbasierten Richtlinien können Sie zulässige oder verweigernde Aktionen und Ressourcen sowie die Bedingungen angeben, unter denen Aktionen zulässig oder verweigert werden. Sie können den Prinzipal nicht in einer identitätsbasierten Richtlinie angeben, da er für

den Benutzer oder die Rolle gilt, dem er zugeordnet ist. Weitere Informationen zu allen Elementen, die Sie in einer JSON Richtlinie verwenden können, finden Sie in der [Referenz zu den IAM JSON Richtlinienelementen](#) im IAMBenutzerhandbuch.

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 anhängen. Beispiele für ressourcenbasierte Richtlinien sind IAM Rollenvertrauensrichtlinien und Amazon S3 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 AWS-Services.

Um den kontoübergreifenden Zugriff zu ermöglichen, können Sie in einer ressourcenbasierten Richtlinie ein ganzes Konto oder IAM Entitäten in einem anderen Konto als Prinzipal angeben. Durch das Hinzufügen eines kontoübergreifenden Auftraggebers zu einer ressourcenbasierten Richtlinie ist nur die halbe Vertrauensbeziehung eingerichtet. Wenn sich der Prinzipal und die Ressource unterscheiden AWS-Konten, muss ein IAM Administrator des vertrauenswürdigen Kontos auch der Prinzipalität (Benutzer oder Rolle) die Erlaubnis erteilen, auf die Ressource zuzugreifen. Sie erteilen Berechtigungen, indem Sie der juristischen Stelle eine identitätsbasierte Richtlinie anfügen. Wenn jedoch eine ressourcenbasierte Richtlinie Zugriff auf einen Prinzipal in demselben Konto gewährt, ist keine zusätzliche identitätsbasierte Richtlinie erforderlich. Weitere Informationen finden Sie [IAMim IAMBenutzerhandbuch unter Kontoübergreifender Ressourcenzugriff](#).

Politische Maßnahmen für HealthImaging

Unterstützt Richtlinienaktionen: Ja

Administratoren können verwenden AWS JSONRichtlinien, um festzulegen, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das `Action` Element einer JSON Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie die zugehörigen AWS API-Betrieb. Es gibt einige Ausnahmen, z. B. Aktionen, für die nur Berechtigungen erforderlich sind und für die es keine entsprechende Operation gibt. API Es gibt auch einige Operationen, die mehrere Aktionen in einer Richtlinie erfordern. Diese zusätzlichen Aktionen werden als abhängige Aktionen bezeichnet.

Schließen Sie Aktionen in eine Richtlinie ein, um Berechtigungen zur Durchführung der zugeordneten Operation zu erteilen.

Eine Liste der HealthImaging Aktionen finden Sie unter [Aktionen definiert von AWS HealthImaging](#) in der Serviceautorisierungsreferenz.

Bei Richtlinienaktionen wird vor der Aktion das folgende Präfix HealthImaging verwendet:

```
AWS
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

```
"Action": [  
  "AWS:action1",  
  "AWS: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 Folgendes verwenden AWS JSONRichtlinien, um festzulegen, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das `Resource` JSON Richtlinienelement gibt das Objekt oder die Objekte an, für die die Aktion gilt. Anweisungen müssen entweder ein `Resource` oder ein `NotResource`-Element enthalten. Es hat sich bewährt, eine Ressource mit ihrem [Amazon-Ressourcennamen \(ARN\)](#) anzugeben. Sie können

dies für Aktionen tun, die einen bestimmten Ressourcentyp unterstützen, der als Berechtigungen auf Ressourcenebene bezeichnet wird.

Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, z. B. Auflistungsoperationen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Eine Liste der HealthImaging Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter [Ressourcentypen definiert von AWS HealthImaging](#) in der Service Authorization Reference. Informationen darüber, mit welchen Aktionen und Ressourcen Sie eine verwenden könnenARN, finden Sie unter [Aktionen definiert von AWS 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 Folgendes verwenden AWS JSONRichtlinien, um festzulegen, wer Zugriff auf was hat. Das heißt, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das Element Condition (oder Condition block) ermöglicht Ihnen die Angabe der Bedingungen, unter denen eine Anweisung wirksam ist. Das Element Condition ist optional. 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.

Wenn Sie mehrere Condition Elemente in einer Anweisung oder mehrere Schlüssel in einem einzelnen Condition Element angeben, AWS wertet sie mithilfe einer logischen AND Operation aus. Wenn Sie mehrere Werte für einen einzelnen Bedingungsschlüssel angeben, AWS wertet die Bedingung mithilfe einer logischen OR Operation aus. Alle Bedingungen müssen erfüllt werden, bevor die Berechtigungen der Anweisung gewährt werden.

Sie können auch Platzhaltervariablen verwenden, wenn Sie Bedingungen angeben. Sie können einem IAM Benutzer beispielsweise nur dann Zugriff auf eine Ressource gewähren, wenn

sie mit seinem IAM Benutzernamen gekennzeichnet ist. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter [IAMRichtlinienelemente: Variablen und Tags](#).

AWS unterstützt globale Bedingungsschlüssel und dienstspezifische Bedingungsschlüssel. Um alle zu sehen AWS globale Bedingungsschlüssel finden Sie unter [AWS Kontexttasten für globale Bedingungen](#) im IAMBenutzerhandbuch.

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 [Aktionen definiert von AWS HealthImaging](#).

Beispiele für HealthImaging identitätsbasierte Richtlinien finden Sie unter [Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging](#)

ACLsin HealthImaging

UnterstütztACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLsähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON Richtliniendokumentformat.

RBACmit HealthImaging

Unterstützt RBAC

Ja

Das in verwendete traditionelle Autorisierungsmodell IAM wird als rollenbasierte Zugriffskontrolle () RBAC bezeichnet. RBACdefiniert Berechtigungen auf der Grundlage der beruflichen Funktion einer Person, die außerhalb von bekannt ist AWS als Rolle. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter [Vergleich ABAC zum herkömmlichen RBAC Modell](#).

ABACmit HealthImaging

Unterstützungen ABAC (Tags in Richtlinien): Teilweise

⚠ Warning

ABAC wird durch die `SearchImageSets` API Aktion nicht erzwungen. Jeder, der Zugriff auf die `SearchImageSets` Aktion hat, kann auf alle Metadaten für Bilddatensätze in einem Datenspeicher zugreifen.

ℹ Note

Bildsätze sind eine untergeordnete Ressource von Datenspeichern. Um verwendet zu werden ABAC, muss ein Bildsatz dasselbe Tag wie ein Datenspeicher haben. Weitere Informationen finden Sie unter [Ressourcen taggen mit AWS HealthImaging](#).

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen auf der Grundlage von Attributen definiert werden. In AWS, diese Attribute werden Tags genannt. Sie können Tags an IAM Entitäten (Benutzer oder Rollen) und an viele andere anhängen AWS Ressourcen schützen. Das Markieren von Entitäten und Ressourcen ist der erste Schritt von ABAC. Anschließend entwerfen Sie ABAC Richtlinien, die Operationen zulassen, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt, auf die er zugreifen möchte.

ABAC ist hilfreich in Umgebungen, die schnell wachsen, und hilft in Situationen, in denen die Richtlinienverwaltung umständlich wird.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungelement 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 finden Sie ABAC unter [Was ist? ABAC](#) im IAM Benutzerhandbuch. Ein Tutorial mit Schritten zur Einrichtung finden Sie im ABAC Benutzerhandbuch unter [Verwenden der attributbasierten Zugriffskontrolle \(ABAC\)](#). IAM

Verwenden temporärer Anmeldeinformationen mit HealthImaging

Unterstützt temporäre Anmeldeinformationen: Ja

Etwas AWS-Services funktioniert nicht, wenn Sie sich mit temporären Anmeldeinformationen anmelden. Für zusätzliche Informationen, einschließlich AWS-Services mit temporären Anmeldeinformationen arbeiten, finden Sie unter [AWS-Services mit denen IAM](#) im IAMBenutzerhandbuch gearbeitet werden kann.

Sie verwenden temporäre Anmeldeinformationen, wenn Sie sich bei der anmelden AWS Management Console mit einer beliebigen Methode außer einem Benutzernamen und einem Passwort. Zum Beispiel, wenn Sie darauf zugreifen AWS Wenn Sie den Single Sign-On-Link (SSO) Ihres Unternehmens verwenden, erstellt dieser Prozess automatisch temporäre Anmeldeinformationen. Sie erstellen auch automatisch temporäre Anmeldeinformationen, wenn Sie sich als Benutzer bei der Konsole anmelden und dann die Rollen wechseln. Weitere Informationen zum Rollenwechsel finden Sie unter [Wechseln zu einer Rolle \(Konsole\)](#) im IAMBenutzerhandbuch.

Sie können temporäre Anmeldeinformationen manuell erstellen, indem Sie AWS CLI or AWS API. Sie können dann diese temporären Anmeldeinformationen für den Zugriff verwenden AWS. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter [Temporäre Sicherheitsanmeldeinformationen unter IAM](#).

Serviceübergreifende Prinzipalberechtigungen für HealthImaging

Unterstützt Forward-Access-Sitzungen (FAS): Ja

Wenn Sie einen IAM Benutzer oder eine Rolle verwenden, um Aktionen auszuführen in 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 für eine Aktion zusätzliche abhängige Aktionen in einer Richtlinie erforderlich sind, finden Sie AWS HealthImaging in der Serviceautorisierungsreferenz unter [Aktionen, Ressourcen und Bedingungsschlüssel für](#).

Servicerollen für HealthImaging

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine [IAMRolle](#), die ein Dienst übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM Administrator kann eine Servicerolle von innen heraus erstellen, ändern und löschenIAM. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen an ein AWS-Service](#) im IAM-Benutzerhandbuch.

⚠ Warning

Das Ändern der Berechtigungen für eine Servicerolle kann zu HealthImaging Funktionseinschränkungen führen. 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 einem verknüpft ist AWS-Service. Der Dienst kann die Rolle übernehmen, eine Aktion in Ihrem Namen durchzuführen. Mit Diensten verknüpfte Rollen erscheinen in Ihrem AWS-Konto und gehören dem Dienst. Ein IAM Administrator kann die Berechtigungen für dienstbezogene Rollen anzeigen, aber nicht bearbeiten.

Einzelheiten zum Erstellen oder Verwalten von dienstbezogenen Rollen finden Sie unter [AWS Dienste, die mit IAM](#) funktionieren. Suchen Sie in der Tabelle nach einem Service mit einem Yes in der Spalte Service-linked role (Serviceverknüpfte Rolle). Wählen Sie den Link Yes (Ja) aus, um die Dokumentation für die serviceverknüpfte Rolle für diesen Service anzuzeigen.

Beispiele für identitätsbasierte Richtlinien für AWS HealthImaging

Standardmäßig sind Benutzer und Rollen nicht berechtigt, HealthImaging Ressourcen zu erstellen oder zu ändern. Sie können auch keine Aufgaben mit dem ausführen AWS Management Console, AWS Command Line Interface (AWS CLI), oder AWS API. Um Benutzern die Erlaubnis zu erteilen, Aktionen mit den Ressourcen durchzuführen, die sie benötigen, kann ein IAM Administrator IAM Richtlinien erstellen. Der Administrator kann dann die IAM Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen übernehmen.

Informationen zum Erstellen einer IAM identitätsbasierten Richtlinie anhand dieser JSON Beispieldokumente finden Sie unter [IAMRichtlinien erstellen](#) im IAMBenutzerhandbuch.

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 Fantastisch](#) in der Service Authorization Reference.

Themen

- [Bewährte Methoden für Richtlinien](#)

- [Verwenden der HealthImaging Konsole](#)
- [Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer](#)

Bewährte Methoden für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand HealthImaging Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Diese Aktionen können Kosten für Sie verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Fangen Sie an mit AWS verwaltete Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Um zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren, verwenden Sie AWS verwaltete Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren. Sie sind in Ihrem AWS-Konto. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie Folgendes definieren AWS vom Kunden verwaltete Richtlinien, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter [AWS verwaltete Richtlinien](#) oder [AWS verwaltete Richtlinien für Jobfunktionen](#) im IAMBenutzerhandbuch.
- Berechtigungen mit den geringsten Rechten anwenden — Wenn Sie Berechtigungen mit IAM Richtlinien festlegen, gewähren Sie nur die Berechtigungen, die für die Ausführung 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 IAM zum Anwenden von Berechtigungen finden Sie [IAMim Benutzerhandbuch unter Richtlinien und Berechtigungen](#). IAM
- Verwenden Sie Bedingungen in IAM Richtlinien, um den Zugriff weiter einzuschränken — Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen einzuschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um anzugeben, dass alle Anfragen mit gesendet werden müssenSSL. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese über eine bestimmte AWS-Service, wie beispielsweise AWS CloudFormation. Weitere Informationen finden Sie unter [IAMJSONRichtlinienelemente: Zustand](#) im IAMBenutzerhandbuch.
- Verwenden Sie IAM Access Analyzer, um Ihre IAM Richtlinien zu validieren, um sichere und funktionale Berechtigungen zu gewährleisten. IAM Access Analyzer validiert neue und bestehende Richtlinien, sodass die Richtlinien der IAM Richtlinienensprache (JSON) und den IAM bewährten Methoden entsprechen. IAMAccess Analyzer bietet mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen, um Sie bei der Erstellung sicherer und funktionaler Richtlinien zu

unterstützen. Weitere Informationen finden Sie unter [IAMAccess Analyzer-Richtlinienvvalidierung](#) im IAMBenutzerhandbuch.

- Multi-Faktor-Authentifizierung erforderlich (MFA) — Wenn Sie ein Szenario haben, in dem IAM Benutzer oder ein Root-Benutzer in Ihrem AWS-Konto, schalten Sie MFA für zusätzliche Sicherheit ein. Wenn Sie festlegen möchten, MFA wann API Operationen aufgerufen werden, fügen Sie MFA Bedingungen zu Ihren Richtlinien hinzu. Weitere Informationen finden Sie unter [Konfiguration des MFA -geschützten API Zugriffs](#) im IAMBenutzerhandbuch.

Weitere Informationen zu bewährten Methoden finden Sie unter [Bewährte Sicherheitsmethoden IAM im IAM](#) Benutzerhandbuch. IAM

Verwenden der HealthImaging Konsole

Um auf die AWS HealthImaging Konsole zugreifen zu können, benötigen Sie ein Mindestmaß an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den HealthImaging Ressourcen in Ihrem AWS-Konto. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die restriktiver ist als die erforderlichen Mindestberechtigungen, funktioniert die Konsole für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie nicht wie vorgesehen.

Sie müssen Benutzern, die nur Anrufe tätigen, keine Mindestberechtigungen für die Konsole gewähren AWS CLI oder das AWS API. Erlauben Sie stattdessen nur den Zugriff auf die Aktionen, die dem API Vorgang entsprechen, den sie ausführen möchten.

Um sicherzustellen, dass Benutzer und Rollen die HealthImaging Konsole weiterhin verwenden können, fügen Sie auch das HealthImaging *ConsoleAccess* oder hinzu *ReadOnly* AWS verwaltete Richtlinie für die Entitäten. Weitere Informationen finden Sie im [Benutzerhandbuch unter Hinzufügen von Berechtigungen für einen IAM](#) Benutzer.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

Dieses Beispiel zeigt, wie Sie eine Richtlinie erstellen könnten, die es IAM Benutzern ermöglicht, die Inline-Richtlinien und verwalteten Richtlinien einzusehen, die mit ihrer Benutzeridentität verknüpft sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe von AWS CLI or AWS API.

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

```

    "Sid": "ViewOwnUserInfo",
    "Effect": "Allow",
    "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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": "*"
}
]
}

```

AWSverwaltete Richtlinien für AWS HealthImaging

Eine von AWS verwaltete Richtlinie ist eine eigenständige Richtlinie, die von AWS erstellt und verwaltet wird. Von AWS verwaltete Richtlinien stellen Berechtigungen für viele häufige Anwendungsfälle bereit, damit Sie beginnen können, Benutzern, Gruppen und Rollen Berechtigungen zuzuweisen.

Beachten Sie, dass AWS-verwaltete Richtlinien möglicherweise nicht die geringsten Berechtigungen für Ihre spezifischen Anwendungsfälle gewähren, da sie für alle AWS-Kunden verfügbar sind.

Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie [kundenverwaltete Richtlinien](#) definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind.

Die Berechtigungen, die in den von AWS verwalteten Richtlinien definiert sind, können nicht geändert werden. Wenn AWS Berechtigungen aktualisiert, die in einer von AWS verwalteten Richtlinie definiert werden, wirkt sich das Update auf alle Prinzipalidentitäten (Benutzer, Gruppen und Rollen) aus, denen die Richtlinie zugeordnet ist. AWS aktualisiert am wahrscheinlichsten eine von AWS verwaltete Richtlinie, wenn ein neuer AWS-Service gestartet wird oder neue API-Operationen für bestehende Services verfügbar werden.

Weitere Informationen finden Sie unter [Von AWS verwaltete Richtlinien](#) im IAM-Benutzerhandbuch.

Themen

- [AWSverwaltete Richtlinie: AWSHealthImagingFullAccess](#)
- [AWSverwaltete Richtlinie: AWSHealthImagingReadOnlyAccess](#)
- [HealthImaging Aktualisierungen der verwalteten AWS Richtlinien](#)

AWSverwaltete Richtlinie: AWSHealthImagingFullAccess

Sie können die `AWSHealthImagingFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt Administratorberechtigungen für alle HealthImaging Aktionen.

```
{
  "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"
            }
        }
    }
]
}

```

AWSverwaltete Richtlinie: AWSHealthImagingReadOnlyAccess

Sie können die `AWSHealthImagingReadOnlyAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt bestimmten HealthImaging AWS-Aktionen nur Leseberechtigungen.

```

{
  "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.

Änderung	Beschreibung	Datum
HealthImaging hat begonnen, Änderungen zu verfolgen	HealthImaging hat begonnen, Änderungen für die AWS verwalteten Richtlinien zu verfolgen.	19. Juli 2023

Fehlerbehebung für AWS HealthImaging -Identität und -Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit HealthImaging und auftreten können. IAM

Themen

- [Ich bin nicht berechtigt, eine Aktion durchzuführen in HealthImaging](#)
- [Ich bin nicht berechtigt, iam auszuführen: PassRole](#)
- [Ich möchte Leute außerhalb meines AWS-Konto um auf meine HealthImaging Ressourcen zuzugreifen](#)

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 mateojackson IAM Benutzer versucht, die Konsole zu verwenden, um Details zu einer fiktiven *my-example-widget* Ressource anzuzeigen, aber nicht über die fiktiven 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 AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht berechtigt sind, die `iam:PassRole` Aktion auszuführen, müssen Ihre Richtlinien aktualisiert werden, damit Sie eine Rolle an HealthImaging diese Person übergeben können.

Etwas 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 AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Leute außerhalb meines AWS-Konto um auf meine HealthImaging Ressourcen zuzugreifen

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden 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 [Wie AWS HealthImaging funktioniert mit IAM](#)
- Um zu erfahren, wie Sie Zugriff auf Ihre Ressourcen gewähren können AWS-Konten die Ihnen gehören, finden Sie unter [Gewähren des Zugriffs für einen IAM Benutzer in einem anderen AWS-Konto die Sie besitzen, finden Sie](#) im IAMBenutzerhandbuch.
- Um zu erfahren, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, siehe Zugriff [gewähren auf AWS-Konten Eigentum Dritter](#) im IAMBenutzerhandbuch.
- Informationen zur [Bereitstellung des Zugriffs über einen Identitätsverbund finden Sie im Benutzerhandbuch unter Zugriff für extern authentifizierte Benutzer \(Identitätsverbund\)](#). IAM
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontenübergreifenden Zugriff finden Sie [IAMim Benutzerhandbuch unter Kontoübergreifender Ressourcenzugriff](#). IAM

Konformitätsvalidierung für AWS HealthImaging

Externe Prüfer bewerten die Sicherheit und Konformität von AWS im HealthImaging Rahmen mehrerer AWS Compliance-Programme. Denn HealthImaging dazu gehört auch HIPAA.

Eine Liste der AWS-Services im Bereich bestimmter Compliance-Programme finden Sie unter [AWS-Services im Bereich nach Compliance-Programm](#). Allgemeine Informationen finden Sie unter [AWS-Compliance-Programme](#).

Sie können Auditberichte von Drittanbietern unter AWS Artifact herunterladen. Weitere Informationen finden Sie unter [Herunterladen von Berichten 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 stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- [AWSPartnerlösungen](#) — 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 zur Erstellung einer Architektur mit HIPAA-konformer Sicherheit und Compliance](#) – In diesem Whitepaper wird beschrieben, wie Unternehmen mithilfe von AWS HIPAA-konforme Anwendungen erstellen können.

- [GxP Systems on AWS](#) — Dieses Whitepaper enthält Informationen darüber, wie die Einhaltung und Sicherheit im Zusammenhang mit AWS GxP angegangen wird, und bietet Anleitungen zur Nutzung AWS von Services im Kontext von GxP.
- [AWS-Compliance-Ressourcen](#) – Diese Arbeitsbücher und Leitfäden könnten für Ihre Branche und Ihren Standort interessant sein.
- [Bewertung von Ressourcen anhand von Regeln](#) — AWS Config bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- [AWS Security Hub](#) – Dieser AWS-Service liefert einen umfassenden Überblick über den Sicherheitsstatus in AWS. So können Sie die Compliance mit den Sicherheitsstandards in der Branche und den bewährten Methoden abgleichen.

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 Ihnen hilft, Ihre AWS Ressourcen zu modellieren und einzurichten, 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 AWS CloudFormation bereitstellt und konfiguriert.

Wenn Sie sie verwenden AWS CloudFormation, können Sie Ihre Vorlage wiederverwenden, um Ihre HealthImaging Ressourcen konsistent und wiederholt einzurichten. Sie beschreiben Ihre Ressourcen dann einmal und können die gleichen Ressourcen dann in mehreren AWS-Konten-Konten und -Regionen immer wieder bereitstellen.

HealthImaging und AWS CloudFormation Vorlagen

Um Ressourcen für und zugehörige Dienste bereitzustellen HealthImaging und zu konfigurieren, müssen Sie sich mit [AWS CloudFormation Vorlagen](#) auskennen. Vorlagen sind formatierte Textdateien in JSON oder YAML. Diese Vorlagen beschreiben die Ressourcen, die Sie in Ihren AWS CloudFormation-Stacks bereitstellen möchten. Wenn Sie noch keine Erfahrungen mit JSON oder YAML haben, können Sie AWS CloudFormation Designer verwenden, der den Einstieg in die Arbeit mit AWS CloudFormation-Vorlagen erleichtert. Weitere Informationen finden Sie unter [Was ist AWS CloudFormation-Designer?](#) im AWS CloudFormation-Benutzerhandbuch.

AWS HealthImaging unterstützt die Erstellung von [Datenspeichern](#) mit AWS 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.

Weitere Informationen zu AWS CloudFormation

Weitere Informationen zu AWS CloudFormation finden Sie in den folgenden Ressourcen.

- [AWS CloudFormation](#)
- [AWS CloudFormation-Benutzerhandbuch](#)
- [AWS 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 basieren auf einer Technologie

[AWS PrivateLink](#), mit der Sie privat auf HealthImaging APIs zugreifen können, ohne dass ein Internet-Gateway, ein NAT-Gerät, eine VPN-Verbindung oder eine AWS Direct Connect-Verbindung erforderlich ist. Instances in Ihrer VPC benötigen keine öffentlichen IP-Adressen, um mit HealthImaging APIs zu kommunizieren. Datenverkehr zwischen Ihrer VPC und HealthImaging verlässt das Amazon-Netzwerk nicht.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere [Elastic-Netzwerk-Schnittstellen](#) in Ihren Subnetzen dargestellt.

Weitere Informationen finden Sie unter [Interface VPC Endpoints \(AWS PrivateLink\)](#) im Amazon VPC-Benutzerhandbuch.

Themen

- [Überlegungen zu HealthImaging VPC-Endpunkten](#)
- [Erstellen eines Schnittstellen-VPC-Endpunkts für HealthImaging](#)
- [Erstellen einer VPC-Endpunkttrichtlinie 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 Ihrem VPC-Endpunkt eine Endpunktrichtlinie hinzufügen, die den Zugriff darauf 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](#).

- 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 `inputOwnerAccountId` bestätigt der Besitzer des Datenspeichers, 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 globale AWS-Infrastruktur ist um AWS-Regionen und Availability Zones herum aufgebaut. AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die mit einem Netzwerk mit geringer Latenz, hohem Durchsatz und hoher Redundanz 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.

Weitere Informationen über AWS-Regionen und Availability Zones finden Sie unter [AWS Globale Infrastruktur](#).

Zusätzlich zur AWS globalen Infrastruktur HealthImaging bietet AWS mehrere Funktionen zur Unterstützung Ihrer Datenausfallsicherheit und Backup-Anforderungen.

HealthImaging AWS-Referenzmaterial

Das folgende Referenzmaterial ist für AWS verfügbar HealthImaging.

Note

Alle HealthImaging Aktionen und Datentypen befinden sich in einer separaten Referenz. Weitere Informationen finden Sie in der [HealthImaging AWS-API-Referenz](#).

Themen

- [DICOMUnterstützung für AWS HealthImaging](#)
- [Überprüfung AWS HealthImaging AWS-Pixeldaten](#)
- [HTJ2K-Dekodierungsbibliotheken für AWS HealthImaging](#)
- [AWS HealthImaging Endpunkte und Kontingente](#)
- [AWS HealthImaging Drosselungsgrenzen](#)
- [AWS HealthImaging Beispielprojekte](#)
- [Verwendung HealthImaging mit einem AWS SDK](#)

DICOMUnterstützung 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 HealthImaging Übertragungssyntaxen und DICOM Elementbeschränkungen entsprechen.

Note

AWS HealthImaging unterstützt derzeit keine Pixeldaten mit binärer Segmentierung oder Bildsequenz aus Symbolen.

Themen

- [Unterstützte Klassen SOP](#)

- [Normalisierung der Metadaten](#)
- [Unterstützte Übertragungssyntaxen](#)
- [DICOMElementeinschränkungen](#)
- [DICOMEinschränkungen bei Metadaten](#)

Unterstützte Klassen SOP

[Mit AWS HealthImaging können Sie DICOM P10 Service-Object Pair \(SOP\) -Instanzen importieren, die mit einer beliebigen SOP Klasse codiert sindUID, einschließlich stillgelegter und privater Instanzen.](#) Alle privaten Attribute werden ebenfalls beibehalten.

Normalisierung der Metadaten

Wenn Sie Ihre DICOM P10-Daten in importieren AWS HealthImaging, werden sie in [Bilddatensätze](#) 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 [DICOMPS3.6 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 [Verzeichnis 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 im [Register der DICOM Datenelemente](#).

AWS HealthImaging unterstützt die folgenden Elemente auf Serienebene:

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 importiert DICOM SOP P10-Instances, die mit den in der folgenden Tabelle aufgeführten Übertragungssyntaxen codiert sind. Zusätzlich zum Speichern der SOP Instanz HealthImaging transkodiert es [Bildrahmen](#) (Pixeldaten) in vier SOP Instanzen, die mit den folgenden HTJ2K Übertragungssyntaxen codiert sind:

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	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	JPEGBaseline (Prozess 1): Standardübertragungssyntax für die verlustbehaftete 8-Bit-Bildkomprimierung JPEG
1.2.840.10008.1.2.4.51	JPEGBaseline (Prozesse 2 und 4): Standardübertragungssyntax für die verlustbehaftete 12-Bit-Bildkomprimierung (nur Prozess 4) JPEG

Übertragungssyntax UID	Name der Übertragungssyntax
1.2.840.10008.1.2.4.57	JPEGVerlustfrei, nicht hierarchisch (Prozess 14)
1.2.840.10008.1.2.4.70	JPEGVerlustfreie, nichthierarchische Vorhersage erster Ordnung (Prozesse 14 [Auswahlwert 1]): Standardübertragungssyntax für die verlustfreie Bildkomprimierung JPEG
1.2.840.10008.1.2.4.80	JPEG-LS Verlustfreie Bildkomprimierung
1.2.840.10008.1.2.4.81	JPEG-LS Verlustbehaftete (nahezu verlustfreie) Bildkomprimierung
1.2.840.10008.1.2.4.90	JPEG2000 Bildkomprimierung (nur verlustfrei)
1.2.840.10008.1.2.4.91	JPEG2000 Bildkomprimierung
1.2.840.10008.1.2.4.201	Bildkomprimierung mit hohem Durchsatz 2000 (nur verlustfrei) JPEG
1.2.840.10008.1.2.4.202	High-Throughput 2000 mit Optionen JPEG zur Bildkomprimierung (nur Lossless) RPCL
1.2.840.10008.1.2.4.203	2000 Bildkomprimierung mit hohem Durchsatz JPEG
1.2.840.10008.1.2.5	RLEVerlustfrei

DICOMElementeinschränkungen

Beim Import Ihrer medizinischen Bilddaten in AWS HealthImaging werden Beschränkungen für die maximale Länge auf die folgenden DICOM Elemente angewendet. Um einen erfolgreichen Import zu erreichen, stellen Sie sicher, dass Ihre Daten die maximale Länge nicht überschreiten.

DICOMElementbeschränkungen beim Import

HealthImaging Schlüsselwort	DICOMSchlüsselwort	DICOM-Schlüssel	Längenbeschränkung
DICOMPatientName	Name des Patienten	(0010,0010)	min: 0, maximal: 256
DICOMPatientID	Patienten-ID	(0010,0020)	min: 0, maximal: 256
DICOMPatientBirthDate	Patientin BirthDate	(0010,0030)	min: 0, maximal: 18
DICOMPatientSex	PatientSex	(0010.0040)	min: 0, maximal: 16
DICOMStudyInstanceUID	StudyInstanceUID	(0020.000 D)	min: 0, maximal: 256
DICOMStudyId	Studien-ID	(0020,0010)	min: 0, maximal: 16
DICOMStudyDescription	StudyDescription	(0008.1030)	min: 0, maximal: 64
DICOMNumberOfStudyRelatedSeries	Anzahl der Reihen, die sich auf die Studie beziehen	(0020, 1206)	min: 0, maximal: 1.000.000
DICOMNumberOfStudyRelatedInstances	NumberOfStudyRelatedInstances	(0020.1208)	min: 0, maximal: 10.000
DICOMAccessionNumber	AccessionNumber	(0008.0050)	min: 0, maximal: 256
DICOMStudyDate	StudyDate	(0008.0020)	min: 0, maximal: 18
DICOMStudyTime	StudyTime	(0008.0030)	min: 0, maximal: 28

DICOMEinschränkungen bei Metadaten

Wenn Sie `UpdateImageSetMetadata` die Aktualisierung von HealthImaging [Metadatenattributen](#) verwenden, werden die folgenden DICOM Einschränkungen angewendet.

- Private Attribute auf Attributen auf der Ebene Patient/Studie/Serie/Instanz können nicht aktualisiert oder entfernt werden, es sei denn, die Aktualisierungsbeschränkung gilt sowohl für `updateableAttributes` `removableAttributes`
- Die folgenden AWS HealthImaging generierten Attribute können nicht aktualisiert werden: `SchemaVersion`,,,,,,, `DatastoreID` `ImageSetID` `PixelData` `Checksum` `Width` `Height` `MinPixelValue` `MaxPixelValue` `FrameSizeInBytes`
- Die folgenden DICOM Attribute können nur aktualisiert werden, wenn das `force` Flag gesetzt ist: `Tag.PixelDataTag.StudyInstanceUID`, `Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID`, `Tag.StudyID`
- Attribute mit dem 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.

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-Decodierungsbibliotheken 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. Mit dem CRC32-Algorithmus wird für jedes Bild eine eindeutige, unveränderliche Prüfsumme für die Bildauflösung (IFRC) generiert. Ein IFRC wird pro Auflösungsstufe für die Pixeldaten in jedem Bild berechnet. Die IFRC-Prüfsummenwerte werden im Metadatendokument (`job-output-manifest.json`) dargestellt und in einer Liste von der Basisauflösung bis zur vollen Auflösung sortiert.
- Nachdem die Bilder in einen HealthImaging [Datenspeicher](#) importiert und in [Bilddatensätze umgewandelt wurden, werden die HTJ2K-codierten Bildframes](#) sofort dekodiert und neue IFRCs berechnet. HealthImaging vergleicht dann die IFRCs der Originalbilder mit voller Auflösung mit den neuen IFRCs der 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 `ImageDataChecksumFromBaseToFullResolution` enthält die IFRC (Prüfsumme) pro Auflösungsstufe. Im Folgenden finden Sie ein Beispiel für die Metadatenausgabe für den IFRC, die im Rahmen des Importauftrags generiert und aufgezeichnet wird. `job-output-manifest.json`

```
"ImageFrames": [{
  "ID": "67890678906789012345123451234512",
  "ImageDataChecksumFromBaseToFullResolution": [
    {
      "Width": 128,
      "Height": 128,
      "Checksum": 2928338830
    },
    {
```

```
    "Width": 256,  
    "Height": 256,  
    "Checksum": 1362274918  
  },  
  {  
    "Width": 512,  
    "Height": 512,  
    "Checksum": 2510355201  
  }  
]
```

- Um Pixeldaten zu verifizieren, rufen Sie das Verfahren [zur Überprüfung der Pixeldaten](#) auf GitHub und folgen Sie den Anweisungen in der README .md Datei, um die fehlerfreie Bildverarbeitung durch [HTJ2K-Decodierungsbibliotheken](#) die verschiedenen, von verwendeten Komponenten unabhängig zu überprüfen. HealthImaging Da die Daten pro Auflösungsstufe schrittweise geladen werden, können Sie den IFRC für Rohdaten selbst berechnen und ihn mit dem IFRC-Wert vergleichen, der in den HealthImaging Metadaten für dieselbe Auflösung angegeben ist, um die Pixeldaten zu verifizieren.

HTJ2K-Dekodierungsbibliotheken für AWS HealthImaging

Während des [Imports](#) HealthImaging codiert AWS alle [Bildrahmen](#) (Pixeldaten) im verlustfreien Format High-Throughput JPEG 2000 (HTJ2K), um eine gleichbleibend schnelle Bildanzeige und universellen Zugriff auf die erweiterten Funktionen von HTJ2K zu ermöglichen. Da Bildrahmen beim Import in HTJ2K codiert werden, müssen sie vor der Anzeige in einem Bildbetrachter dekodiert werden.

Note

HTJ2K ist in [Teil 15 des JPEG2000-Standards \(ISO/IEC 15444-15:2019\)](#) definiert. HTJ2K behält die erweiterten Funktionen von JPEG2000 wie Auflösungskalierbarkeit, Bezirke, Kachelung, hohe Bittiefe, mehrere Kanäle und Farbraumunterstützung bei.

Themen

- [HTJ2K-Decodierungsbibliotheken](#)
- [Bildbetrachter](#)

HTJ2K-Decodierungsbibliotheken

[Abhängig von Ihrer Programmiersprache empfehlen wir die folgenden Dekodierungsbibliotheken, um Bildrahmen zu dekodieren.](#)

- [NVIDIA nvJPEG2000](#) — 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](#)

AWS HealthImaging Endpunkte und Kontingente

Die folgenden Themen enthalten Informationen zu AWS HealthImaging Dienstendpunkten und Kontingenten.

Themen

- [Service-Endpunkte](#)
- [Servicekontingente](#)

Service-Endpunkte

Ein Dienstendpunkt URL identifiziert einen Host und einen Port als Einstiegspunkt für einen Webdienst. 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 Dienstendpunkte für aufgeführt. AWS HealthImaging

Name der Region	Region	Endpunkt	Protocol (Protokoll)
USA Ost (Nord-Virginia)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS
USA West (Oregon)	us-west-2	medical-imaging.us-west-2.amazonaws.com	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

Wenn Sie HTTP Anfragen zum Aufrufen von AWS HealthImaging Aktionen verwenden, müssen Sie je nach aufgerufenen Aktionen unterschiedliche Endpunkte verwenden. Im folgenden Menü sind die verfügbaren Dienstendpunkte für HTTP Anfragen und die von ihnen unterstützten Aktionen aufgeführt.

Unterstützte API Aktionen für Anfragen HTTP

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
- StartDICOMImport Job
- GetDICOMImport Job
- ListDICOMImport Arbeitsplätze
- 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 WADO -RS-Dienste. Weitere Informationen finden Sie unter [DICOMDaten werden abgerufen von HealthImaging](#).

Die folgenden DICOMweb Dienste sind über den Endpunkt zugänglich:

```
https://dicom-medical-imaging.region.amazonaws.com
```

- GetDICOMInstance
- GetDICOMInstanceMetadata
- GetDICOMInstanceFrames

Servicekontingente

Dienstkontingente sind als 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 aufgeführt AWS HealthImaging.

Name	Standard	Anpas	Beschreibung
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

Name	Standard	Anpas	Beschreibung
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 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 AWS Region

Name	Standard	Anpas	Beschreibung
Maximale Anzahl verschachtelter Ordner in einem DICOM Importauftrag	Jede unterstützte Region: 10 000	Nein	Die maximale Anzahl verschachtelter Ordner in einem DICOM Importauftrag in der aktuellen Region AWS
Die maximale Größenbeschränkung für Payloads (in KB) wurde akzeptiert von UpdateImageSetMetadata	Jede unterstützte Region: 10 Kilobyte	Ja	Die maximale Nutzlastgrößenbeschränkung (in KB), die von der aktuellen UpdateImageSetMetadata Region akzeptiert wird AWS
Maximale Größe (in GB) aller Dateien in einem DICOM Importauftrag	Jede unterstützte Region: 10 Gigabyte	Nein	Die maximale Größe (in GB) aller Dateien in einem DICOM Importauftrag in der aktuellen AWS Region
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

AWS HealthImaging Drosselungsgrenzen

Für Ihr AWS Konto gelten Drosselungslimits, die für Aktionen gelten. AWS HealthImaging API Bei allen Aktionen wird ein `ThrottlingException` Fehler ausgegeben, wenn die Drosselungsgrenzen überschritten werden. [Weitere Informationen finden Sie in der AWS HealthImaging 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](#).

In der folgenden Tabelle sind die Einschränkungsgrenzen für systemeigene AWS HealthImaging API Aktionen und Dienstdarstellungen aufgeführt. DICOMweb

AWS HealthImaging Drosselungsgrenzwerte

Aktion	Drosselungsrate	Gaspedal geplatzt
CreateDatastore	0,085 Tipps	1 TL
GetDatastore	10 TPS	20 Teelöffel
ListDatastores	5 Tipps	10 TPS
DeleteDatastore	0,085 TL	1 TL
StartDICOMImport Job	0,25 Tipps	10 Teelöffel
GetDICOMImport Job	25 Tipps	50 Teelöffel
ListArbeitsplätze istDICOMImport	10 TPS	20 Tipps
SearchImageSets	25 Teelöffel	50 Teelöffel
GetImageSet	25 Teelöffel	50 Teelöffel
GetImageSetMetadata	50 Teelöffel	100 Teelöffel

Aktion	Drosselungsrate	Gaspedal geplatzt
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
G * etDICOMInstance	50 Tipps	100 Teelöffel
G Metadaten* etDICOMInstance	50 Tipps	100 Teelöffel
G-Rahmen* etDICOMInstance	50 Tipps	100 Teelöffel

*Darstellung einer Dienstleistung DICOMweb

AWS HealthImaging Beispielprojekte

AWS HealthImaging bietet die folgenden Beispielprojekte für GitHub.

[DICOMAufnahme von vor Ort 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, in die importiert werden soll, in eine Warteschlange. AWS HealthImaging Sie besteht aus einer Komponente, die am Edge läuft und von der Cloud verwaltet wird, und einer Datenerfassungspipeline [AWS IoT Greengrass](#), die in DICOM der Cloud läuft.

AWS

[Proxy mit Markierung auf Kachelebene \(\) TLM](#)

Ein [AWS Cloud Development Kit \(AWS CDK\)](#) Projekt zum Abrufen von Bildrahmen mithilfe AWS HealthImaging von Markierungen auf Kachelebene (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 und bereitstellt. GET Standardmäßig authentifiziert der Endpunkt Anfragen mit einem Amazon Cognito JSON Cognito-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.

[AWS HealthImaging Viewer-Benutzeroberfläche](#)

Ein [AWS Amplify](#) Projekt zur Bereitstellung einer Frontend-Benutzeroberfläche mit Backend-Authentifizierung, mit der Sie Metadatenattribute und Bildrahmen (Pixeldaten) von Bildsätzen anzeigen können, die AWS HealthImaging mithilfe der progressiven Dekodierung gespeichert sind. Sie können optional die oben genannten Projekte Tile Level Marker (TLM) Proxy und/oder Amazon CloudFront Delivery integrieren, um Bildrahmen mit einer alternativen Methode zu laden.

[AWS HealthImaging DICOMwebProxy](#)

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 [Verwenden DICOMweb mit AWS HealthImaging](#)

Weitere Beispielprojekte finden Sie unter [AWS HealthImaging Beispiele](#) für GitHub.

Verwendung HealthImaging mit einem AWS SDK

AWS Software Development Kits (SDKs) sind für viele gängige 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 for C++	AWS SDK for C++ Code-Beispiele
AWS CLI	AWS CLI Codebeispiele
AWS SDK for Go	AWS SDK for Go Codebeispiele
AWS SDK for Java	AWS SDK for Java Codebeispiele
AWS SDK for JavaScript	AWS SDK for JavaScript Codebeispiele
AWS SDK for Kotlin	AWS SDK for Kotlin Codebeispiele
AWS SDK for .NET	AWS SDK for .NET Codebeispiele
AWS SDK for PHP	AWS SDK for PHP Codebeispiele
AWS Tools for PowerShell	Tools für PowerShell Codebeispiele
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) Codebeispiele
AWS SDK for Ruby	AWS SDK for Ruby Codebeispiele
AWS SDK for Rust	AWS SDK for Rust Codebeispiele
AWS SDK für SAP ABAP	AWS SDK für SAP ABAP Codebeispiele
AWS SDK for Swift	AWS SDK for Swift Codebeispiele

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.

AWS HealthImaging Veröffentlichungen

Die folgende Tabelle zeigt, wann Funktionen und Updates für den AWS HealthImaging Service und die Dokumentation veröffentlicht wurden. Weitere Informationen zu einer Version finden Sie im verlinkten Thema.

Änderung	Beschreibung	Datum
Kehren Sie zur Versions-ID eines früheren Bildsatzes zurück	HealthImaging stellt den <code>revertToVersionId</code> Parameter bereit, mit dem die Versions-ID eines früheren Bildsatzes wiederhergestellt werden kann. Weitere Informationen finden Sie revertToVersionId in der AWS HealthImaging APIReferenz.	24. Juli 2024
Erzwingen Sie die Funktionalität für die Änderung des Bilddatensatzes	HealthImaging stellt den <code>Overrides</code> Datentyp mit dem optionalen <code>forced</code> Anforderungsparameter bereit. Die Einstellung dieses Parameters erzwingt die <code>CopyImageSet</code> Aktionen <code>UpdateImageSetMetadata</code> und, auch wenn Metadaten auf Patienten-, Studien- oder Serienebene nicht übereinstimmen. Weitere Informationen finden Sie in der Referenz unter Überschreibungen . AWS HealthImaging API	24. Juli 2024

- `UpdateImageSetMetadata` 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 AWS HealthImaging APIReferenz.

- `CopyImageSet` Force-Funktionalität — HealthImaging führt den optionalen `force` Anforderungsparameter für das Kopieren von Bilddatensätzen ein. Durch die Einstellung dieses Parameters wird die `CopyImageSet` Aktion erzwungen, auch wenn Metadaten auf Patienten-, Studien- oder Serienebene im `sourceImageSet` und `destinationImageSet` angegeben sind.

onImageSet nicht übereinstimmen. In diesen Fällen bleiben die inkonsistenten Metadaten in der unverändert. destinationImageSet Weitere Informationen finden Sie [CopyImageSet](#) in der AWS HealthImaging APIReferenz.

[Kopieren Sie Teilmengen von Instanzen SOP](#)

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 den GetDICOMInstanceMetadata Dienst zur Rückgabe von Metadaten (.jsonDatei) für DICOM Teil 10 bereit. Weitere Informationen finden Sie unter [DICOMInstanz-Metadaten abrufen von HealthImaging](#).

11. Juli 2024

[GetDICOMInstanceFrames für die Rückgabe von DICOM Instanz-Frames \(Pixeldaten\)](#)

HealthImaging stellt den GetDICOMInstanceFrames Dienst zur Rückgabe von DICOM Part 10-Frames (multipart Anfrage) bereit. Weitere Informationen finden Sie unter [DICOM Instanz-Frames abrufen von HealthImaging](#).

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 [DICOMElementeinschränkungen](#).

28. Juni 2024

- 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 zulässig Acquisition UID:
 - Das erste Element eines beliebigen Elements UID kann Null sein
 - UID kann mit einer oder mehreren führenden Nullen beginnen

- UIDskann bis zu 256 Zeichen lang sein

Ereignis-Benachrichtigungen	HealthImaging lässt sich in Amazon integrieren EventBridge , um ereignisgesteuerte Anwendungen zu unterstützen. Weitere Informationen finden Sie unter Amazon verwenden EventBridge mit HealthImaging .	5. Juni 2024
GetDICOMInstance für die Rückgabe von DICOM Instanzdaten	HealthImaging stellt den GetDICOMInstance Dienst zur Rückgabe von Instanzdaten (. dcmDatei) von DICOM Teil 10 bereit. Weitere Informationen finden Sie unter Eine DICOM Instanz abrufen von HealthImaging .	15. Mai 2024
Kontoübergreifender Import	HealthImaging bietet Unterstützung für Datenimporte aus Amazon S3 S3-Buckets, die sich in anderen unterstützten Regionen befinden. Weitere Informationen finden Sie unter Kontoübergreifender Import für AWS HealthImaging .	15. Mai 2024

Verbesserungen bei der Suche nach Bilddatensätzen

HealthImaging SearchImageSets action unterstützt die folgenden Suchverbesserungen. Weitere Informationen finden Sie unter [Suche nach Bilddatensätzen](#).

3. April 2024

- Zusätzliche Unterstützung für die Suche nach UpdatedAt und SeriesInstanceUID
- Suchen Sie zwischen Startzeit und Endzeit
- Sortiere die Suchergebnisse nach Ascending oder Descending
- DICOMSerienparameter werden in Antworten zurückgegeben

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

[Transfersyntaxen für JPEG Lossless und HTJ2K](#)

HealthImaging unterstützt die folgenden Übertragungssyntaxen für Jobimporte. Weitere Informationen finden Sie unter [Unterstützte Übertragungssyntaxen](#).

16. Februar 2024

- 1.2.840.10008.1.2.4.57 — Verlustfrei, nicht hierarchisch (Prozess 14) JPEG
- 1.2.840.10008.1.2.4.201 — Bildkomprimierung mit hohem JPEG Durchsatz von 2000 (nur verlustfrei)
- 1.2.840.10008.1.2.4.202 — High-Throughput 2000 mit Optionen zur Bildkomprimierung (nur verlustfrei) JPEG RPCL
- 1.2.840.10008.1.2.4.203 — Bildkomprimierung für JPEG 2000 mit hohem Durchsatz

[Getestete Codebeispiele](#)

HealthImaging Die Dokumentation enthält getestete Codebeispiele AWS SDKs für AWS CLI und für Python JavaScript, Java und C++. Weitere Informationen finden Sie unter [Codebeispiele für die HealthImaging Verwendung AWS SDKs](#).

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 Servicekontingente .	1. Dezember 2023
Schnellere Importe	HealthImaging bietet 20-mal schnellere Importe in allen unterstützten Regionen. Weitere Informationen finden Sie unter Service-Endpunkte .	1. Dezember 2023
CloudFormation Unterstützung	HealthImaging unterstützt Infrastructure as Code (IaC) für die Bereitstellung von Datenspeichern. Weitere Informationen finden Sie unter HealthImaging AWS-Ressourcen erstellen mit AWS 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.	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.