개발자 가이드

AWS HealthImaging

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS HealthImaging: 개발자 가이드

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

AWS HealthImaging이란 무엇인가요? 1
중요 공지 사항 2
Features 2
관련 서비스 3
액세스 4
HIPAA 5
요금 5
시작 6
개념 6
데이터 스토어 6
이미지 세트 7
메타데이터 7
이미지 프레임 7
설정 7
에 가입 AWS 계정 8
관리자 액세스 권한이 있는 사용자 생성 8
S3 버킷 생성 10
데이터 스토어 생성 10
IAM 사용자를 생성합니다. 11
IAM 역할 생성 11
설치 AWS CLI 14
자습서 15
데이터 스토어 관리 16
데이터 스토어 생성 16
데이터 스토어 속성 가져오기 23
데이터 스토어 목록 29
데이터 스토어 삭제 36
스토리지 티어 이해 42
이미징 데이터 가져오기 45
가져오기 작업에 대한 이해 45
가져오기 작업 시작 48
가져오기 작업 속성 가져오기 56
가져오기 작업을 나열하기 62
이미지 세트 액세스하기 68

이미지 세트 이해 68
이미지 세트 검색 75
이미지 세트 속성 가져오기 100
이미지 세트 메타데이터 가져오기 105
이미지 세트 픽셀 데이터 가져오기 115
이미지 세트 수정하기 123
이미지 세트 버전 목록 123
이미지 세트 메타데이터 업데이트 130
..... 147
이미지 세트 복사 150
이미지 세트 삭제 164
리소스에 태그 지정 170
리소스에 태그 지정 170
리소스에 대한 태그 나열 175
리소스의 태그 해제 179
코드 예제 185
기본 사항 191
HealthImaging 시작 192
작업 198
시나리오 332
이미지 세트 및 이미지 프레임 시작하기 333
데이터 저장소에 태그 지정 387
이미지 세트 태그 지정 397
DICOMweb 408
데이터 불러오기 408
인스턴스 가져오기 410
인스턴스 메타데이터 가져오기 412
인스턴스 프레임 가져오기 413
시리즈 메타데이터 가져오기 415
데이터 검색 416
HealthImaging용 DICOMweb 검색 APIs 417
HealthImaging에 지원되는 DICOMweb 쿼리 유형 418
연구 검색 420
시리즈 검색 421
인스턴스 검색 422
모니터링 424

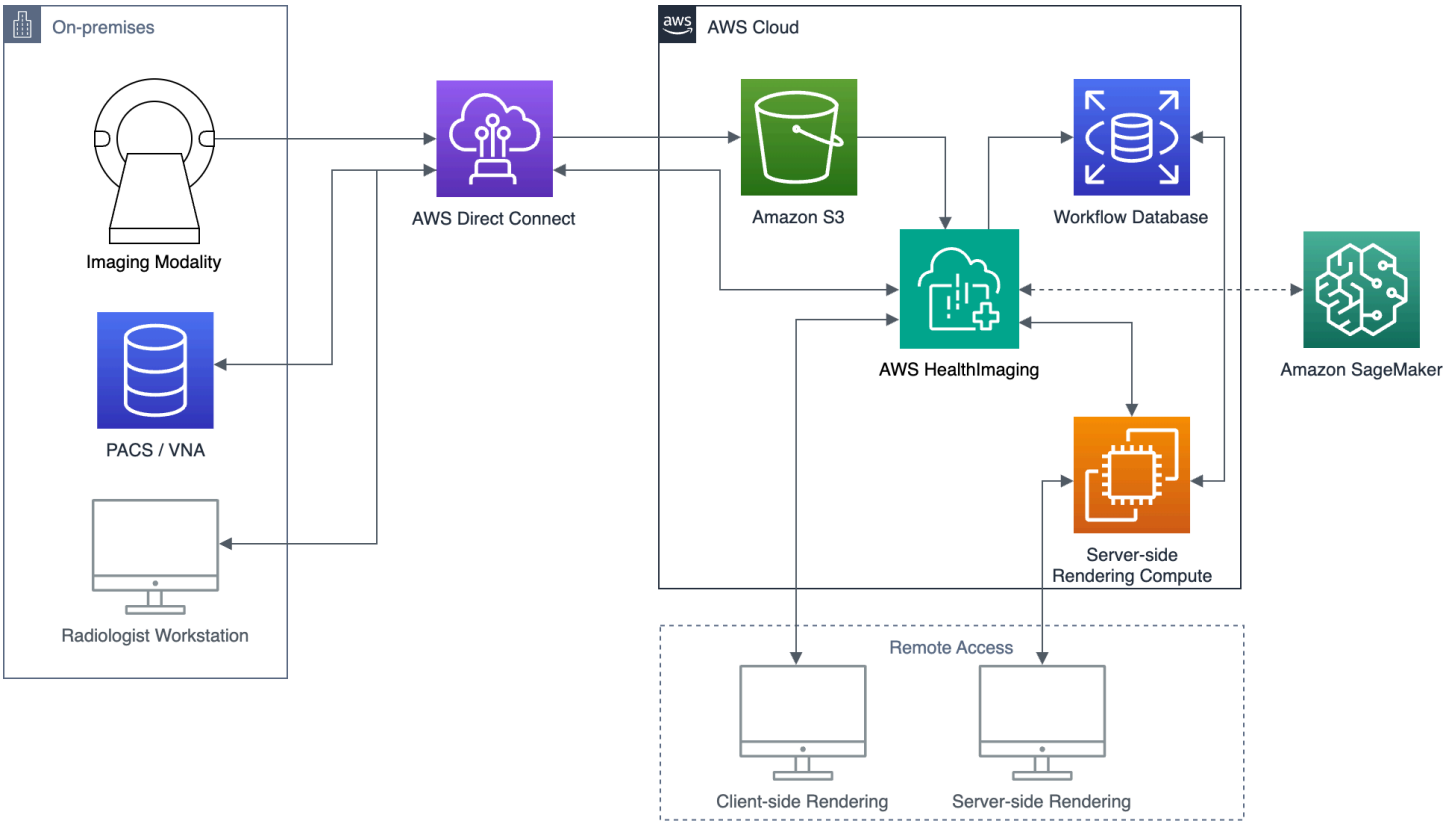
CloudTrail(API 호출) ... 424
추적 생성 ... 425
로그 항목 이해 ... 426
CloudWatch(지표) ... 427
HealthImaging 지표 보기 ... 428
경보 생성 ... 429
EventBridge(이벤트) ... 429
EventBridge로 전송된 HealthImaging 이벤트 ... 429
HealthImaging 이벤트 구조 및 예제 ... 431
보안 ... 447
데이터 보호 ... 448
데이터 암호화 ... 449
네트워크 트래픽 개인 정보 보호 ... 459
자격 증명 및 액세스 관리 ... 459
대상 ... 460
ID를 통한 인증 ... 460
정책을 사용하여 액세스 관리 ... 463
AWS HealthImaging에서 IAM을 사용하는 방법 ... 466
자격 증명 기반 정책 예제 ... 472
AWS 관리형 정책 ... 475
교차 서비스 혼동된 대리자 방지 ... 477
문제 해결 ... 479
규정 준수 확인 ... 480
인프라 보안 ... 481
코드형 인프라 ... 481
HealthImaging 및 AWS CloudFormation 템플릿 ... 482
에 대해 자세히 알아보기 AWS CloudFormation ... 482
VPC 엔드포인트 ... 482
VPC 엔드포인트 고려 사항 ... 483
VPC 엔드포인트 생성 ... 483
VPC 엔드포인트 정책 생성 ... 484
교차 계정 가져오기 ... 485
복원성 ... 486
레퍼런스 ... 488
DICOM ... 488
지원되는 SOP 클래스 ... 489

메타데이터 정규화 489
지원되는 전송 구문 494
DICOM 요소 제약 조건 496
DICOM 메타데이터 제약 조건 497
HealthImaging 498
엔드포인트 및 할당량 499
제한 한계 503
픽셀 데이터 확인 505
HTJ2K 디코딩 라이브러리 507
샘플 프로젝트 508
AWS SDKs 작업 510
출시 512
..... dxxii

AWS HealthImaging이란 무엇인가요?

AWS HealthImaging은 의료 서비스 제공자, 생명과학 조직 및 소프트웨어 파트너가 페타바이트 규모로 클라우드에 의료 이미지를 저장, 분석 및 공유할 수 있도록 지원하는 HIPAA 적격 서비스입니다. HealthImaging 사용 사례는 다음과 같습니다.

- 엔터프라이즈 이미징 - 지연 시간이 짧은 성능과 고가용성을 유지하면서 AWS 클라우드에서 직접 의료 영상 데이터를 저장하고 스트리밍합니다.
- 장기 이미지 아카이브 - 1초 미만의 이미지 검색 액세스를 유지하면서 장기 이미지 아카이브 비용을 절감합니다.
- AI/ML 개발 - 다른 도구 및 서비스의 지원을 통해 이미징 아카이브에 대한 인공 지능 및 기계 학습(AI/ML) 추론을 실행합니다.
- 멀티모달 분석 - 임상 영상 데이터를 AWS HealthLake (상태 데이터) 및 AWS HealthOmics (오믹스 데이터)와 결합하여 정밀 의학에 대한 인사이트를 제공합니다.



AWS HealthImaging은 이미지 데이터(예: X-Ray, CT, MRI, Ultrasound)에 대한 액세스를 제공하므로 클라우드에 내장된 의료 영상 애플리케이션이 이전에는 온프레미스에서만 가능한 성능을 달성할 수

있습니다. HealthImaging을 사용하면 AWS 클라우드에 있는 각 의료 이미지의 신뢰할 수 있는 단일 사본에서 대규모로 의료 영상 애플리케이션을 실행하여 인프라 비용을 절감할 수 있습니다.

주제

- 중요 공지 사항
- AWS HealthImaging의 특징
- 관련 AWS 서비스
- AWS HealthImaging에 액세스
- HIPAA 자격 및 데이터 보안
- 요금

중요 공지 사항

AWS HealthImaging은 전문적인 의학적 조언, 진단 또는 치료를 대체하지 않으며 질병이나 건강 상태를 치료, 완화, 예방 또는 진단하기 위한 것도 아닙니다. AWS HealthImaging을 사용할 경우 인적 검토를 실시할 책임이 있습니다. 예를 들면 임상적 의사 결정을 알리도록 고안된 타사 제품과 연계하는 것도 포함됩니다. AWS HealthImaging은 올바른 의학적 판단을 적용하여 교육을 받은 의료 전문가의 검토를 거친 후 환자 치료 또는 임상 시나리오에서만 사용해야 합니다.

AWS HealthImaging의 특징

AWS HealthImaging은 다음 기능을 제공합니다.

개발자에게 친숙한 DICOM 메타데이터

AWS HealthImaging은 DICOM 메타데이터를 개발자 친화적인 형식으로 반환하여 애플리케이션 개발을 간소화합니다. 이미징 데이터를 가져온 후에는 익숙하지 않은 그룹/요소 16진수 대신 사람이 이해하기 쉬운 키워드를 사용하여 개별 메타데이터 속성에 액세스할 수 있습니다. 환자, 연구 및 시리즈 수준의 DICOM 요소가 정규화되므로 애플리케이션 개발자가 SOP 인스턴스 간의 불일치를 처리하지 않아도 됩니다. 또한 메타데이터 속성 값은 네이티브 런타임 유형에서 직접 액세스할 수 있습니다.

SIMD-가속 이미지 디코딩

AWS HealthImaging은 고급 이미지 압축 코덱인 고처리량 JPEG 2000(HTJ2K) 이미지로 인코딩된 이미지 프레임(픽셀 데이터)을 반환합니다. HTJ2K는 최신 프로세서의 SIMD(단일 명령 다중 데이터)를 활용하여 새로운 수준의 성능을 제공합니다. HTJ2K는 JPEG 2000보다 10배 빠르며 다른 모

든 DICOM 전송 구문보다 2배 이상 빠릅니다. WASM-SIMD를 활용하면 공간을 많이 차지하지 않는 웹 뷰어에 이같은 극한의 속도를 제공할 수 있습니다. 자세한 내용은 지원되는 전송 구문 단원을 참조하십시오.

픽셀 데이터 확인

AWS HealthImaging은 가져오기 중에 모든 이미지의 무손실 인코딩 및 디코딩 상태를 확인하여 내장된 픽셀 데이터 검증을 제공합니다. 자세한 내용은 픽셀 데이터 확인 단원을 참조하십시오.

업계 최고의 성능

AWS HealthImaging은 효율적인 메타데이터 인코딩, 무손실 압축, 점진적 해상도 데이터 액세스 덕분에 이미지 로딩 성능의 새로운 표준을 정립합니다. 효율적인 메타데이터 인코딩을 통해 이미지 뷰어와 AI 알고리즘은 이미지 데이터를 로드하지 않고도 DICOM 연구의 내용을 이해할 수 있습니다. 고급 이미지 압축 덕분에 이미지 품질 저하 없이 이미지를 더 빠르게 로드할 수 있습니다. 점진적 해상도를 사용하면 썸네일, 관심 영역 및 저해상도 모바일 디바이스의 이미지를 훨씬 더 빠르게 로드할 수 있습니다.

확장 가능 DICOM 가져오기

AWS HealthImaging 가져오기는 최신 클라우드 네이티브 기술을 활용하여 여러 DICOM 연구를 병렬로 가져옵니다. 새 데이터에 대한 임상 워크로드에 영향을 주지 않으면서 기록 아카이브를 빠르게 가져올 수 있습니다. 지원되는 SOP 인스턴스 및 전송 구문에 대한 내용은 DICOM 섹션을 참조하십시오.

서비스 관리형 DICOM 데이터 계층 구조

AWS HealthImaging은 환자, 연구 및 시리즈 수준 DICOM 데이터 요소별 가져오기 시 DICOM P10 데이터를 자동으로 구성합니다. 서비스는이 DICOM 데이터를 DICOM 시리즈에 해당하는 이미지 세트로 구성하여 사후 가져오기 워크플로를 간소화합니다. 새 데이터를 가져오면 연구 및 시리즈 수준 조직이 유지됩니다.

DICOMweb API 호환성

AWS HealthImaging은 DICOMweb 준수 APIs 제공하여 통합을 간소화하고 기존 애플리케이션과의 상호 운용성을 지원합니다. 또한이 서비스는 메타데이터 업데이트 작업과 같이 DICOMweb 표준에서 지원하지 않는 작업을 활성화하는 클라우드 네이티브 APIs를 제공합니다.

관련 AWS 서비스

AWS HealthImaging은 다른 AWS 서비스와 긴밀하게 통합됩니다. 다음 서비스에 대한 지식은 HealthImaging을 완전히 활용하는 데 유용합니다.

- AWS Identity and Access Management - IAM을 사용하여 자격 증명과 HealthImaging 리소스에 대한 액세스를 안전하게 관리합니다.
- Amazon Simple Storage Service - Amazon S3를 스테이징 영역으로 사용하여 DICOM 데이터를 HealthImaging으로 가져옵니다.
- Amazon CloudWatch - CloudWatch를 사용하여 HealthImaging 리소스를 관찰하고 모니터링합니다.
- AWS CloudTrail - CloudTrail을 사용하여 HealthImaging 사용자 활동 및 API 사용량을 추적합니다.
- AWS CloudFormation - AWS CloudFormation 를 사용하여 코드형 인프라(IaC) 템플릿을 구현하여 HealthImaging에서 리소스를 생성합니다.
- AWS PrivateLink - Amazon VPC를 사용하여 데이터를 인터넷에 노출하지 않고 HealthImaging과 Amazon Virtual Private Cloud 간에 연결을 설정합니다.
- Amazon EventBridge - EventBridge를 사용하여 HealthImaging 이벤트를 대상으로 라우팅하는 규칙을 생성하여 확장 가능한 이벤트 기반 애플리케이션을 생성합니다.

AWS HealthImaging에 액세스

AWS Management Console AWS Command Line Interface 및SDK를 사용하여 AWS HealthImaging에 액세스할 수 있습니다. AWS SDKs 이 가이드에서는 AWS Management Console 및 SDK의 AWS CLI 및 AWS 코드 예제에 대한 절차 지침을 제공합니다. SDKs

AWS Management Console

는 HealthImaging 및 관련 리소스를 관리하기 위한 웹 기반 사용자 인터페이스를 AWS Management Console 제공합니다. AWS 계정에 가입한 경우 HealthImaging 콘솔에 로그인할 수 있습니다.

AWS Command Line Interface (AWS CLI)

는 다양한 AWS 제품에 대한 명령을 AWS CLI 제공하며 Windows, Mac 및 Linux에서 지원됩니다. 자세한 내용은 AWS Command Line Interface 사용 설명서를 참조하십시오.

AWS SDKs

AWS SDKs 소프트웨어 개발자를 위한 라이브러리, 코드 예제 및 기타 리소스를 제공합니다. 이러한 라이브러리는 요청에 암호화 서명, 요청 재시도, 오류 응답 처리 등과 같은 작업을 자동으로 관리하는 기본 기능을 제공합니다. 자세한 내용은 빌드 기반 도구를 참조하세요 AWS.

HTTP 요청

HTTP 요청을 사용하여 HealthImaging 작업을 호출할 수 있지만 사용 중인 작업 유형에 따라 다른 엔드포인트를 지정해야 합니다. 자세한 내용은 HTTP 요청에 대해 지원되는 API 작업 단원을 참조하십시오.

HIPAA 자격 및 데이터 보안

이것은 HIPAA 적격 서비스입니다. 1996 AWS년 미국 건강 보험 양도 및 책임에 관한 법(HIPAA) 및 AWS 서비스를 사용하여 보호 대상 건강 정보(PHI)를 처리, 저장 및 전송하는 방법에 대한 자세한 내용은 HIPAA 개요를 참조하세요.

PHI 및 개인 식별 정보(PII)가 포함된 HealthImaging에 대한 연결은 암호화되어야 합니다. 기본적으로 HealthImaging에 대한 모든 연결은 TLS를 통한 HTTPS를 사용합니다. HealthImaging은 암호화된 고객 콘텐츠를 저장하고 AWS 공동 책임 모델에 따라 운영됩니다.

규정 준수에 대한 자세한 내용은 AWS HealthImaging의 규정 준수 검증 섹션을 참조하십시오.

요금

HealthImaging은 지능형 계층화를 통해 임상 데이터의 수명 주기 관리를 자동화할 수 있도록 지원합니다. 자세한 내용은 스토리지 티어 이해 단원을 참조하십시오.

일반 요금 정보는 AWS HealthImaging 요금을 참조하십시오. 비용을 추정하려면 AWS HealthImaging 요금 계산기를 참조하십시오.

AWS HealthImaging 시작하기

AWS HealthImaging 사용을 시작하려면 AWS 계정을 설정하고 AWS Identity and Access Management 사용자를 생성합니다. AWS CLI또는 AWS SDK 사용하려면 반드시 이를 설치하고 구성해야 합니다.

HealthImaging 개념에 대해 알아보고 설정한 후 코드 예제가 포함된 간단한 자습서를 사용하여 시작할 수 있습니다.

주제

- AWS HealthImaging의 개념
- AWS HealthImaging 설정
- AWS HealthImaging 튜토리얼

AWS HealthImaging의 개념

다음 용어와 개념은 AWS HealthImaging의 이해와 사용에 매우 중요합니다.

개념

- 데이터 스토어
- 이미지 세트
- 메타데이터
- 이미지 프레임

데이터 스토어

데이터 스토어는 AWS 리전단일 저장소에 있는 의료 영상 데이터의 리포지토리입니다. AWS 계정은 데이터 스토어가 0개이거나 많을 수 있습니다. 데이터 스토어에는 자체 AWS KMS 암호화 키가 있으므로 한 데이터 스토어의 데이터를 다른 데이터 스토어의 데이터와 물리적이고 논리적으로 분리할 수 있습니다. 데이터 스토어는 IAM 역할, 권한 및 속성 기반 액세스 제어를 사용하여 액세스 제어를 지원합니다.

자세한 내용은 데이터 스토어 관리 및 스토리지 티어 이해 섹션을 참조하십시오.

이미지 세트

이미지 세트는 관련 의료 영상 데이터를 최적화하기 위한 추상 그룹화 메커니즘을 정의하는 AWS 개념입니다. DICOM P10 영상 데이터를 AWS HealthImaging 데이터 스토어로 가져오면 메타데이터와 이미지 프레임(픽셀 데이터)으로 구성된 이미지 세트로 변환하는 것입니다. HealthImaging은 연구, 시리즈 및 인스턴스의 DICOM 계층 구조에 따라 가져온 데이터를 구성하려고 시도합니다. HealthImaging 관리형 계층에 성공적으로 추가된 DICOM 인스턴스는 기본 이미지 세트로 표시됩니다. DICOM P10 데이터를 가져오면 새 기본 이미지 세트를 생성하거나, 인스턴스가 이미 기본 컬렉션에 있는 경우 인스턴스를 기존 기본 이미지 세트에 병합하거나, 메타데이터 요소가 충돌하는 경우 새 비기본 이미지 세트를 생성합니다.

자세한 내용은 이미징 데이터 가져오기 및 이미지 세트 이해 섹션을 참조하세요.

메타데이터

메타데이터는 이미지 세트 내에 있는 비픽셀 속성입니다. DICOM의 경우 여기에는 환자 인구 통계, 시술 세부정보 및 기타 획득 관련 매개변수가 포함됩니다. AWS HealthImaging은 애플리케이션이 빠르게 액세스할 수 있도록 이미지 세트를 메타데이터와 이미지 프레임(픽셀 데이터)으로 분리합니다. 이는 픽셀 데이터가 필요하지 않은 이미지 뷰어, 분석 및 AI/ML 사용 사례에 유용합니다. DICOM 데이터는 환자, 연구 및 시리즈 수준에서 정규화되어 불일치를 제거합니다. 이를 통해 데이터 사용이 단순화되고 안전성이 향상되며 액세스 성능이 개선됩니다.

자세한 내용은 이미지 세트 메타데이터 가져오기 및 메타데이터 정규화 섹션을 참조하세요.

이미지 프레임

이미지 프레임은 2D 의료 영상을 구성하기 위해 이미지 세트 내에 존재하는 픽셀 데이터입니다. 일부 파일은 가져오기 중에 원래 전송 구문 인코딩을 유지하는 반면, 다른 파일은 기본적으로 High-Throughput JPEG 2000(HTJ2K) 무손실로 트랜스코딩됩니다. 이미지 프레임이 HTJ2K로 인코딩된 경우 이미지 뷰어에서 보기 전에 디코딩해야 합니다. 자세한 내용은 지원되는 전송 구문, 이미지 세트 픽셀 데이터 가져오기, HTJ2K 디코딩 라이브러리 섹션을 참조하세요.

AWS HealthImaging 설정

AWS HealthImaging을 사용하기 전에 AWS 환경을 설정해야 합니다. 다음 주제는 다음 섹션에 있는 튜토리얼의 사전 요구 사항입니다.

주제

- 에 가입 AWS 계정
- 관리자 액세스 권한이 있는 사용자 생성
- S3 버킷 생성
- 데이터 스토어 생성
- HealthIfig 전체 액세스 권한이 있는 IAM 사용자 생성
- 가져오기용 IAM 역할 생성
- 설치 AWS CLI (선택 사항)

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.

2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

S3 버킷 생성

DICOM P10 데이터를 AWS HealthImaging으로 가져오려면 두 개의 Amazon S3 버킷을 사용하는 것이 좋습니다. Amazon S3 입력 버킷은 가져올 DICOM P10 데이터를 저장하고 HealthImaging은 이 버킷에서 데이터를 읽습니다. Amazon S3 출력 버킷은 가져오기 작업의 처리 결과와 HealthImaging 쓰기 작업을 이 버킷에 저장합니다. 이를 시각적으로 표현하려면 가져오기 작업에 대한 이해의 다이어그램을 참조하세요.

> **Note**
>
> AWS Identity and Access Management (IAM) 정책으로 인해 Amazon S3 버킷 이름은 고유해야 합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 이름 지정 규칙을 참조하세요.

이 안내서의 목적상 가져올 IAM 역할에 다음과 같은 Amazon S3 입력 및 출력 버킷을 지정합니다.

- 입력 버킷: `arn:aws:s3:::`*`amzn-s3-demo-source-bucket`*
- 출력 버킷: `arn:aws:s3:::`*`amzn-s3-demo-logging-bucket`*

자세한 내용은 Amazon S3 사용 설명서의 버킷 생성하기를 참조하세요.

데이터 스토어 생성

의료 영상 데이터를 가져올 때 AWS HealthImaging 데이터 스토어는 변환된 DICOM P10 파일의 결과(이를 이미지 세트라고 함)를 보관합니다. 이를 시각적으로 표현하려면 가져오기 작업에 대한 이해의 다이어그램을 참조하세요.

> **Tip**
>
> `datastoreID`는 데이터 스토어를 생성할 때 생성됩니다. 이 섹션의 뒷부분에 나오는 가져올 trust relationship을(를) 완료할 때는 `datastoreID`를 사용해야 합니다.

데이터 스토어를 생성하려면 데이터 스토어 생성 섹션을 참조하세요.

HealthIfig 전체 액세스 권한이 있는 IAM 사용자 생성

> **모범 사례**
>
> 가져오기, 데이터 액세스, 데이터 관리 등 다양한 요구 사항에 맞게 별도의 IAM 사용자를 생성하는 것이 좋습니다. 이는 AWS Well-Architected 프레임워크의 최소 권한 액세스 부여와 부합합니다.
> 다음 섹션의 튜토리얼에서는 단일 IAM 사용자를 사용하게 됩니다.

IAM 사용자 생성

1. IAM 사용 설명서의 AWS 계정에서 IAM 사용자 생성 지침을 따릅니다. 명확화를 사용자 `ahiadmin`(또는 유사한 이름)의 이름을 지정하는 것을 고려하십시오.
2. `AWSHealthImagingFullAccess` 관리형 정책을 IAM 사용자에 배정합니다. 자세한 내용은 AWS 관리형 정책: AWSHealthImagingFullAccess 단원을 참조하십시오.

> **Note**
>
> IAM 권한 범위를 좁힐 수 있습니다. 자세한 내용은 AWS AWS HealthImaging에 대한 관리형 정책 단원을 참조하십시오.

가져오기용 IAM 역할 생성

> **Note**
>
> 다음 지침은 DICOM 데이터를 가져오기 위해 Amazon S3 버킷에 대한 읽기 및 쓰기 액세스 권한을 부여하는 AWS Identity and Access Management (IAM) 역할을 참조합니다. 다음 섹션의 튜토리얼에는 역할이 필요하지만 정책을 직접 작성하는 것보다 사용하기 쉽기 때문에 AWS AWS HealthImaging에 대한 관리형 정책을(를) 이용하여 사용자, 그룹 및 역할에 IAM 권한을 추가하는 것이 좋습니다.

IAM 역할은 계정에 생성할 수 있는, 특정 권한을 지닌 IAM 자격 증명입니다. 가져오기 작업을 시작하려면 `StartDICOMImportJob` 작업을 호출하는 IAM 역할을 DICOM P10 데이터를 읽고 가져오기 작업

처리 결과를 저장하는 데 사용되는 Amazon S3 버킷에 대한 액세스 권한을 부여하는 사용자 정책에 연결해야 합니다. 또한 AWS HealthImaging이 역할을 맡을 수 있는 신뢰 관계(정책)를 할당해야 합니다.

가져오기용 IAM 역할을 생성하려면

1. IAM 콘솔을 사용하여 `ImportJobDataAccessRole`(이)라는 이름의 역할을 생성합니다. 다음 섹션의 튜토리얼에서는 이 역할을 사용합니다. 자세한 내용은 IAM 사용 설명서의 IAM 역할 생성을 참조하세요.

   > **Tip**
   > 이 설명서의 목적을 위해 가져오기 작업 시작의 코드 예제는 `ImportJobDataAccessRole` IAM 역할을 참조하세요.

2. IAM 권한 정책을 IAM 역할에 연결합니다. 이 권한 정책은 Amazon S3 입력 및 출력 버킷에 대한 액세스 권한을 부여합니다. IAM 역할 `ImportJobDataAccessRole`에 다음의 IAM 권한 정책을 연결합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-source-bucket",
                "arn:aws:s3:::amzn-s3-demo-logging-bucket"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-source-bucket/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
```

```
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-logging-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

3. 다음 신뢰 관계(정책)을 `ImportJobDataAccessRole` IAM 역할에 연결합니다. 신뢰 정책에는 섹션 데이터 스토어 생성 작성 시 생성된 `datastoreId`가 필요합니다. 이 주제 다음에 나오는 튜토리얼에서는 AWS HealthImaging 데이터 스토어 하나를 사용하지만 데이터 스토어별 Amazon S3 버킷, IAM 역할 및 신뢰 정책을 사용한다고 가정합니다.

> **Note**
>
> 이 신뢰 정책의 `Condition` 블록은 특정 AWS HealthImaging 데이터 스토어에만 액세스할 수 있도록 하여 혼동된 대리자 문제를 방지하는 데 도움이 됩니다. 이 보안 조치에 대한 자세한 내용은 HealthImaging의 교차 서비스 혼동된 대리자 방지를 참조하세요.

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

```
    ]
}
```

AWS HealthImaging으로 IAM 정책을 생성하고 사용하는 방법에 대한 자세한 내용은 AWS HealthImaging 자격 증명 및 액세스 관리 섹션을 참조하세요.

일반적인 IAM 역할에 대한 자세한 내용은 IAM 사용 설명서의 IAM 역할 섹션을 참조하세요. 일반적인 IAM 정책 및 권한에 대한 자세한 내용은 IAM 사용 설명서의 IAM 정책 및 권한을 참조하세요.

설치 AWS CLI (선택 사항)

AWS Command Line Interface를 사용하는 경우 다음 절차가 필요합니다. AWS Management Console AWS SDKs를 사용하는 경우 다음 절차를 건너뛸 수 있습니다.

를 설정하려면 AWS CLI

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은AWS Command Line Interface 사용 설명서에서 다음 주제를 참조하세요.

   - 최신 버전의 설치 또는 업데이트 AWS CLI
   - 시작하기 AWS CLI

2. AWS CLI `config` 파일에 관리자의 명명된 프로필을 추가합니다. AWS CLI 명령을 실행할 때 이 프로파일을 사용합니다. 최소 권한이라는 보안 원칙에 따라, 수행 중인 작업과 관련된 권한을 가진 별도의 IAM 역할을 생성하는 것이 좋습니다. 명명된 프로파일에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 구성 및 자격 증명 파일 설정을 참조하세요.

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

3. 다음 `help` 명령을 사용하여 설정을 확인합니다.

   ```
   aws medical-imaging help
   ```

   AWS CLI 가 올바르게 구성된 경우 AWS HealthImaging에 대한 간략한 설명과 사용 가능한 명령 목록이 표시됩니다.

AWS HealthImaging 튜토리얼

목표

이 자습서의 목적은 DICOM P10 바이너리(`.dcm` 파일)를 AWS HealthImaging 데이터 스토어로 가져와 메타데이터와 이미지 프레임(픽셀 데이터)으로 구성된 이미지 세트로 변환하는 것입니다. ??? DICOM 데이터를 가져온 후 HealthImaging 클라우드 네이티브 작업을 사용하여 액세스 기본 설정에 따라 이미지 세트, 메타데이터 및 이미지 프레임에 액세스합니다.

사전 조건

이 튜토리얼을 완료하려면 설정에 나열된 모든 절차가 필요합니다.

튜토리얼 단계

1. 가져오기 작업 시작
2. 가져오기 작업 속성 가져오기
3. 이미지 세트 검색
4. 이미지 세트 속성 가져오기
5. 이미지 세트 메타데이터 가져오기
6. 이미지 세트 픽셀 데이터 가져오기
7. 데이터 스토어 삭제

AWS HealthImaging으로 데이터 스토어 관리

AWS HealthImaging을 사용하면 의료 이미지 리소스용 데이터 스토어를 생성하고 관리할 수 있습니다. 다음 주제에서는 HealthImaging 클라우드 네이티브 작업을 사용하여 AWS Management Console AWS CLI및 AWS SDKs를 사용하여 데이터 스토어를 생성, 설명, 나열 및 삭제하는 방법을 설명합니다.

> **Note**
>
> 이 장의 마지막 주제는 스토리지 티어를 이해하는 것입니다. 의료 영상 데이터를 HealthImaging 데이터 스토어로 가져오면 시간과 사용량에 따라 두 스토리지 계층 간에 자동으로 이동합니다. 스토리지 계층의 요금 수준은 다르므로 결제 목적으로 인식되는 계층 이동 프로세스와 HealthImaging 리소스를 이해하는 것이 중요합니다.

주제

- 데이터 스토어 생성
- 데이터 스토어 속성 가져오기
- 데이터 스토어 목록
- 데이터 스토어 삭제
- 스토리지 티어 이해

데이터 스토어 생성

`CreateDatastore` 작업을 사용하여 DICOM P10 파일을 가져오기 위한 AWS HealthImaging 데이터 스토어를 생성합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `CreateDatastore` 섹션을 참조하세요.

> **중요**
>
> 개인 정보(PHI), 개인 식별 정보(PII) 혹은 기타 기밀 정보 또는 민감한 정보를 데이터 스토어 이름으로 지정하지 마세요.

데이터 스토어 생성

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 생성 페이지를 엽니다.
2. 세부 정보의 데이터 스토어 이름에서 데이터 스토어의 이름을 입력합니다.
3. 데이터 암호화에서 리소스를 암호화할 AWS KMS 키를 선택합니다. 자세한 내용은 AWS HealthImaging의 데이터 보호 단원을 참조하십시오.
4. 태그 - 선택 사항에서 데이터 스토어를 생성할 때 데이터 스토어에 태그를 추가할 수 있습니다. 자세한 내용은 리소스에 태그 지정 단원을 참조하십시오.
5. 데이터 스토어 생성을 선택합니다.

AWS CLI 및 SDKs

Bash

AWS CLI Bash 스크립트 사용

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

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

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

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

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

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

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

  local error_code=${?}
```

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

  echo "$response"

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 CreateDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 생성

다음 `create-datastore` 코드 예시에서는 `my-datastore`라는 데이터 스토어를 생성합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 생성을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CreateDatastore를 참조하세요.

Java

SDK for Java 2.x

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

       return "";
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 CreateDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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

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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CreateDatastore를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

데이터 스토어 속성 가져오기

`GetDatastore` 작업을 사용하여 AWS HealthImaging 데이터 스토어 속성을 검색합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `GetDatastore` 섹션을 참조하세요.

데이터 스토어 속성 가져오기

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열립니다. 세부정보 섹션에서 모든 데이터 스토어 속성을 사용할 수 있습니다. 관련 이미지 세트, 가져오기, 태그를 보려면 해당 탭을 선택하세요.

AWS CLI 및 SDKs

Bash

AWS CLI Bash 스크립트 사용

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

###############################################################################
# function imaging_get_datastore
#
# Get a data store's properties.
#
# Parameters:
#       -i data_store_id - The ID of the data store.
#
```

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

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

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

  local response
```

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

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

  echo "$response"

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 GetDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 속성 가져오기

다음 `get-datastore` 코드 예시에서는 데이터 스토어 속성을 가져옵니다.

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

출력:

```
{
    "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"
    }
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 속성 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetDatastore를 참조하세요.

Java

SDK for Java 2.x

```
    public static DatastoreProperties
 getMedicalImageDatastore(MedicalImagingClient medicalImagingClient,
            String datastoreID) {
        try {
            GetDatastoreRequest datastoreRequest = GetDatastoreRequest.builder()
                    .datastoreId(datastoreID)
                    .build();
            GetDatastoreResponse response =
 medicalImagingClient.getDatastore(datastoreRequest);
            return response.datastoreProperties();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxx:datastore/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response.datastoreProperties;
```

```
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetDatastore를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

데이터 스토어 목록

`ListDatastores` 작업을 사용하여 AWS HealthImaging에서 사용 가능한 데이터 스토어를 나열합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `ListDatastores` 섹션을 참조하세요.

데이터 스토어 목록을 만들려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

- HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.

  모든 데이터 스토어는 데이터 스토어 섹션 아래에 나열됩니다.

AWS CLI 및 SDKs

Bash

AWS CLI Bash 스크립트 사용

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

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

  # Retrieve the calling parameters.
  while getopts "h" option; do
    case "${option}" in
      h)
        usage
        return 0
        ;;
```

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

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

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

  echo "$response"

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 나열

다음 `list-datastores` 코드 예시에서는 사용 가능한 데이터 스토어를 나열합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListDatastores를 참조하세요.

Java

SDK for Java 2.x

```
    public static List<DatastoreSummary>
 listMedicalImagingDatastores(MedicalImagingClient medicalImagingClient) {
        try {
            ListDatastoresRequest datastoreRequest =
 ListDatastoresRequest.builder()
                    .build();
            ListDatastoresIterable responses =
 medicalImagingClient.listDatastoresPaginator(datastoreRequest);
            List<DatastoreSummary> datastoreSummaries = new ArrayList<>();

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

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

```
            System.exit(1);
        }

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page.datastoreSummaries);
    console.log(page);
  }
  // {
```

```
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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

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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListKeyspaces를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

데이터 스토어 삭제

`DeleteDatastore` 작업을 사용하여 AWS HealthImaging 데이터 스토어를 삭제합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `DeleteDatastore` 섹션을 참조하세요.

> **Note**
>
> 데이터 스토어를 삭제하려면 먼저 데이터 스토어에 있는 모든 이미지 세트를 삭제해야 합니다. 자세한 내용은 이미지 세트 삭제 단원을 참조하십시오.

데이터 스토어를 삭제하려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.
3. Delete(삭제)를 선택합니다.

   데이터 스토어 삭제 페이지가 열립니다.
4. 데이터 스토어 삭제를 확인하려면 텍스트 입력 필드에 데이터 스토어 이름을 입력합니다.
5. 데이터 스토어 삭제를 선택합니다.

AWS CLI 및 SDKs

Bash

AWS CLI Bash 스크립트 사용

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

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

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

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

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

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

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

  local error_code=${?}

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

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 삭제

다음 `delete-datastore` 코드 예시에서는 데이터 스토어를 삭제합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteDatastore를 참조하세요.

Java

SDK for Java 2.x

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteDatastore를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

스토리지 티어 이해

AWS HealthImaging은 자동 임상 수명 주기 관리에 지능형 계층화를 사용합니다. 그 결과 신규 또는 활성 데이터와 장기 보관 데이터 모두에 대해 마찰 없이 뛰어난 성능과 가격을 제공합니다. HealthImaging은 다음 등급에 따라 GB/월 당 스토리지 요금을 청구합니다.

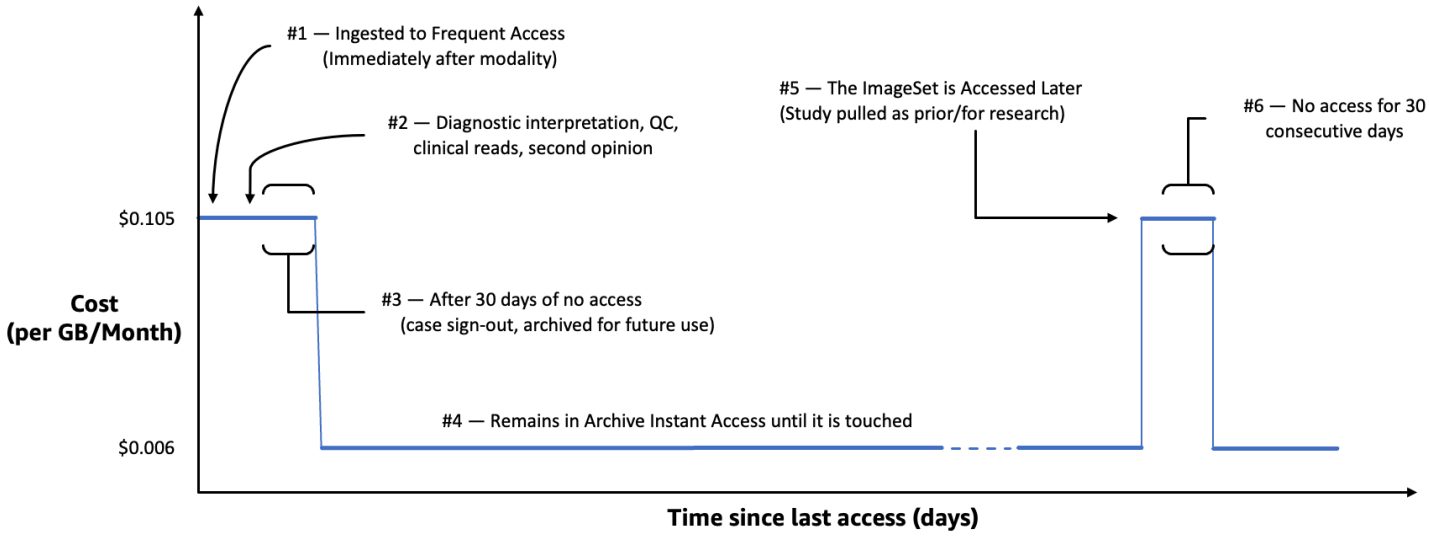
- 자주 액세스 티어 - 자주 액세스하는 데이터용 티어입니다.
- 아카이브 즉시 액세스 티어 - 아카이브된 데이터용 티어입니다.

> **Note**
>
> Frequent Access 계층과 Archive Instant Access 계층 간에는 성능 차이가 없습니다. 지능형 계층화는 특정 이미지 세트 API 작업에 적용됩니다. 지능형 티어는 데이터 스토어, 가져오기 및 API 작업 태깅을 인식하지 못합니다. 티어 간 이동은 API 사용량에 따라 자동으로 수행되며 다음 섹션에 설명되어 있습니다.

티어 이동은 어떻게 작동하나요?

- 가져오기 후 이미지 세트는 자주 액세스 티어에서 시작됩니다.
- 연속 30일 동안 아무 작업도 하지 않으면 이미지 세트는 자동으로 아카이브 즉시 액세스 티어로 이동합니다.
- 아카이브 즉시 액세스 티어의 이미지 세트는 수정된 후에만 자주 액세스 티어로 다시 이동합니다.

다음 그래프는 HealthImaging 지능형 계층화 프로세스의 개요를 제공합니다.



터치로 간주되는 것은 무엇입니까?

터치는 AWS Management Console AWS CLI또는 AWS SDKs이며 다음과 같은 경우에 발생합니다.

1. 새 이미지 세트가 생성될 때(`StartDICOMImportJob`또는`CopyImageSet`)
2. 이미지 세트가 업데이트될 때 (`UpdateImageSetMetadata`또는`CopyImageSet`)
3. 이미지 세트의 결합된 메타데이터 또는 이미지 프레임(픽셀 데이터)을 읽을 때 (`GetImageSetMetaData`또는`GetImageFrame`)

다음과 같은 HealthImaging API 작업을 수행하면 터치가 발생하고 이미지 세트를 아카이브 인스턴트 액세스 티어에서 자주 액세스 티어로 이동합니다.

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

> **Note**
>
> 이미지 프레임(픽셀 데이터)은 `UpdateImageSetMetadata` 작업을 사용하여 삭제할 수 없지만 여전히 청구 대상입니다.

다음과 같은 HealthImaging API 작업은 터치로 간주되지 않습니다. 따라서 아카이브 즉시 액세스 티어에서 자주 액세스 티어로 이미지 세트를 이동하지 않습니다.

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

AWS HealthImaging을 사용한 이미징 데이터 가져오기

가져오기 작업은 의료 영상 데이터를 Amazon S3 입력 버킷에서 AWS HealthImaging 데이터 스토어로 이동하는 프로세스입니다. 가져오기 도중에 AWS HealthImaging은 DICOM P10 파일을 메타데이터와 이미지 프레임(픽셀 데이터)으로 구성된 이미지 세트로 변환하기 전에 픽셀 데이터 검증 검사를 수행합니다.

> **중요**
>
> HealthImaging 가져오기 작업은 DICOM 인스턴스 바이너리(`.dcm` 파일)를 처리하고 이미지 세트로 변환합니다. HealthImaging 클라우드 네이티브 작업(APIs)을 사용하여 데이터 스토어 및 이미지 세트를 관리합니다. HealthImaging의 DICOMweb 서비스 표현을 사용하여 DICOMweb 응답을 반환합니다.

다음 주제에서는 AWS Management Console AWS CLI및 AWS SDKs를 사용하여 의료 영상 데이터를 HealthImaging 데이터 스토어로 가져오는 방법을 설명합니다.

주제

- 가져오기 작업에 대한 이해
- 가져오기 작업 시작
- 가져오기 작업 속성 가져오기
- 가져오기 작업을 나열하기

가져오기 작업에 대한 이해

AWS HealthImaging에서 데이터 스토어를 생성한 후에는 Amazon S3 입력 버킷에서 데이터 스토어로 의료 영상 데이터를 가져와서 이미지 세트를 생성해야 합니다. AWS Management Console AWS CLI, 및 AWS SDKs를 사용하여 가져오기 작업을 시작, 설명 및 나열할 수 있습니다.

DICOM P10 데이터를 AWS HealthImaging 데이터 스토어로 가져오면 서비스는 메타데이터 요소를 기반으로 연구 UID, 시리즈 UID, 인스턴스 UID의 DICOM 계층 구조에 따라 인스턴스를 자동으로 구성하려고 시도합니다. 가져온 데이터의 메타데이터 요소가 데이터 스토어의 기존 기본 이미지 세트와 충돌하지 않는 경우 가져온 데이터가 기본으로 설정됩니다. 새로 가져온 DICOM P10 데이터의 메타데이터 요소가 기존 기본 이미지 세트와 충돌하는 경우 새 데이터가 기본이 아닌 이미지 세트에 추가됩니다. 데이터 가져오기가 기본이 아닌 이미지 세트를 생성하면 AWS

HealthImagingsuccess.ndjson은를 사용하여 EventBridge 이벤트를 내보내고 `isPrimary: False`에 기록된 레코드도 `importResponse` 객체 `isPrimary: False` 내에 있습니다.

데이터를 가져올 때 HealthImaging은 다음을 수행합니다.

- DICOM 시리즈로 구성된 인스턴스를 하나의 가져오기 작업으로 가져오고 인스턴스가 데이터 스토어에 이미 있는 인스턴스와 충돌하지 않는 경우 모든 인스턴스는 하나의 기본 이미지 세트로 구성됩니다.
- DICOM 시리즈를 구성하는 인스턴스를 둘 이상의 가져오기 작업으로 가져오고 인스턴스가 데이터 스토어에 이미 있는 인스턴스와 충돌하지 않는 경우 모든 인스턴스는 하나의 기본 이미지 세트로 구성됩니다.
- 인스턴스를 두 번 이상 가져오면 최신 버전이 기본 이미지 세트에 저장된 이전 버전을 덮어쓰고 기본 이미지 세트의 버전 번호가 증가합니다.

이미지 세트 메타데이터 업데이트에 설명된 단계를 사용하여 기본 인스턴스를 업데이트할 수 있습니다.

Amazon S3에서 HealthImaging 데이터 스토어로 의료 영상 파일을 가져올 때는 다음 사항에 유의하세요.

- DICOM 시리즈에 해당하는 인스턴스는 기본으로 표시된 단일 이미지 세트에 자동으로 결합됩니다.
- DICOM P10 데이터를 하나의 가져오기 작업 또는 여러 가져오기 작업으로 가져올 수 있으며, 서비스는 인스턴스를 DICOM 시리즈에 해당하는 기본 이미지 세트로 구성합니다.
- 가져오기 중에는 특정 DICOM 요소에 길이 제약이 적용됩니다. 가져오기 작업을 성공적으로 수행하려면 의료 영상 데이터가 길이 제한을 초과하지 않도록 확인하십시오. 자세한 내용은 DICOM 요소 제약 조건 단원을 참조하십시오.
- 픽셀 데이터 확인 검사는 가져오기 작업을 시작할 때 수행됩니다. 자세한 내용은 픽셀 데이터 확인 단원을 참조하십시오.
- HealthImaging 가져오기 작업에는 엔드포인트, 할당량 및 스로틀링 제한이 있습니다. 자세한 내용은 엔드포인트 및 할당량 및 제한 한계 섹션을 참조하세요.
- 각 가져오기 작업의 처리 결과는 `outputS3Uri` 위치에 저장됩니다. 처리 결과는 `job-output-manifest.json` 파일 `SUCCESS` 및 `FAILURE` 폴더로 구성됩니다.

> **Note**
>
> 단일 가져오기 작업에 최대 1만 개의 중첩된 폴더를 포함할 수 있습니다.

- 이 job-output-manifest.json 파일에는 처리된 데이터에 대한 jobSummary 출력 및 추가 세부 정보가 들어 있습니다. 다음 예제 출력은 job-output-manifest.json 파일을 보여줍니다.

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

- SUCCESS 폴더에는 성공적으로 가져온 모든 이미징 파일의 결과가 들어 있는 success.ndjson 파일이 들어 있습니다. 다음 예제 출력은 success.ndjson 파일을 보여줍니다.

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

```
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105630.1.0.1.dcm","importResponse
{"imageSetId":"1234561234561234567891789101 2", "isPrimary": True}}
```

- FAILURE 폴더에는 성공적으로 가져오지 못한 모든 이미징 파일의 결과가 들어 있는 failure.ndjson 파일이 들어 있습니다. 다음 예제 출력은 failure.ndjson 파일을 보여줍니다.

```
{"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"}}
```

- 가져오기 작업은 90일 동안 작업 목록에 보관된 후 보관됩니다.

가져오기 작업 시작

StartDICOMImportJob 작업을 사용하여 픽셀 데이터 확인 검사를 시작하고 AWS HealthImaging 데이터 스토어로 대량 데이터 가져오기를 시작합니다. 가져오기 작업은 inputS3Uri 파라미터로 지정된 Amazon S3 입력 버킷에 있는 DICOM P10 파일을 가져옵니다. 가져오기 작업 처리 결과는 outputS3Uri 파라미터로 지정된 Amazon S3 출력 버킷에 저장됩니다.

> **Note**
>
> 가져오기 작업을 시작하기 전에 다음 사항에 유의하세요.
>
> - HealthImaging은 전송 구문이 다른 DICOM P10 파일 가져오기를 지원합니다. 일부 파일은 가져오기 중에 원래 전송 구문 인코딩을 유지하는 반면, 다른 파일은 기본적으로 HTJ2K 무손실로 트랜스코딩됩니다. 자세한 내용은 지원되는 전송 구문 단원을 참조하십시오.
> - HealthImaging은 지원되는 다른 리전에 위치한 Amazon S3 버킷에서 데이터 가져오기를 지원합니다. 이 기능을 수행하려면 가져오기 작업을 시작할 때 inputOwnerAccountId 파라미터를 제공합니다. 자세한 내용은 에 대한 교차 계정 가져오기 AWS HealthImaging 단원을 참조하십시오.
> - HealthImaging은 가져오는 동안 특정 DICOM 요소에 길이 제약 조건을 적용합니다. 자세한 내용은 DICOM 요소 제약 조건 단원을 참조하십시오.

다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 및 코드 예제에 AWS CLI 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `StartDICOMImportJob` 섹션을 참조하세요.

가져오기 작업을 시작하려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.
3. DICOM 데이터 가져오기를 선택합니다.

   DICOM 데이터 가져오기 페이지가 열립니다.
4. 세부 정보 섹션에 다음 정보를 입력합니다.
   - 이름(선택 사항)
   - S3에서 소스 위치 가져오기
   - 소스 버킷 소유자의 계정 ID(선택 사항)
   - 암호화 키(선택 사항)
   - S3의 출력 대상
5. 서비스 액세스 섹션에서 기존 서비스 역할 사용을 선택하고 서비스 역할 이름 메뉴에서 역할을 선택하거나 새 서비스 역할 생성 및 사용을 선택합니다.
6. 가져오기를 선택합니다.

AWS CLI 및 SDKs

C++

SDK for C++

```
//! Routine which starts a HealthImaging import job.
```

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

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

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

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

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 StartDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

dicom 가져오기 작업 시작

다음 `start-dicom-import-job` 코드 예시에서는 dicom 가져오기 작업을 시작합니다.

```
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"
```

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 가져오기 작업 시작을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 StartDICOMImportJob을 참조하세요.

Java

SDK for Java 2.x

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

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

       return "";
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 StartDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 StartDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 StartTranscriptionJob을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

가져오기 작업 속성 가져오기

`GetDICOMImportJob` 작업을 사용하여 AWS HealthImaging 가져오기 작업 속성에 대해 자세히 알아봅니다. 예를 들어, 가져오기 작업을 시작한 후 `GetDICOMImportJob`(을)를 실행하여 작업 상태를 찾을 수 있습니다. `COMPLETED`로 `jobStatus`가 반환되면 이미지 세트에 액세스할 준비가 된 것입니다.

> **Note**
>
> `jobStatus`는 가져오기 작업의 실행을 나타냅니다. 따라서 가져오기 프로세스 중에 검증 문제가 발견되더라도 가져오기 작업은 `jobStatus`를 `COMPLETED`로 반환할 수 있습니다. `jobStatus`가 `COMPLETED`로 반환되는 경우, 개별 P10 객체 가져오기의 성공 또는 실패에 대한 세부 정보를 제공하는 Amazon S3에 작성된 출력 매니페스트를 검토하는 것이 좋습니다.

다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 및 코드 예제에 AWS CLI 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `GetDICOMImportJob` 섹션을 참조하세요.

가져오기 작업 속성을 가져오려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열립니다. 이미지 세트 탭은 기본적으로 선택됩니다.
3. 가져오기 탭을 선택합니다.
4. 가져오기 작업을 선택합니다.

   가져오기 작업 세부 정보 페이지가 열리고 가져오기 작업에 대한 속성이 표시됩니다.

AWS CLI 및 SDKs

C++

SDK for C++

```
//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
  \param clientConfig: Aws client configuration.
  \return GetDICOMImportJobOutcome: The import job outcome.
*/
Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
AwsDoc::Medical_Imaging::getDICOMImportJob(const Aws::String &dataStoreID,
                                           const Aws::String &importJobID,
                                           const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetDICOMImportJobRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetJobId(importJobID);
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome outcome =
 client.GetDICOMImportJob(
            request);
    if (!outcome.IsSuccess()) {
        std::cerr << "GetDICOMImportJob error: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome;
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 GetDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

dicom 가져오기 작업의 속성 가져오기

다음 `get-dicom-import-job` 코드 예시에서는 dicom 가져오기 작업의 속성을 가져옵니다.

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

출력:

```
{
    "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/"
    }
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 가져오기 작업 속성 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetDICOMImportJob을 참조하세요.

Java

SDK for Java 2.x

```
    public static DICOMImportJobProperties getDicomImportJob(MedicalImagingClient
 medicalImagingClient,
            String datastoreId,
```

```
            String jobId) {

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

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
```

```
  jobId = "xxxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetDICOMImportJob을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

가져오기 작업을 나열하기

`ListDICOMImportJobs` 작업을 사용하여 특정 HealthImaging 데이터 스토어에 대해 생성된 가져오기 작업을 나열합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `ListDICOMImportJobs` 섹션을 참조하세요.

> **Note**
>
> 가져오기 작업은 90일 동안 작업 목록에 보관된 후 보관됩니다.

가져오기 작업을 나열하려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열립니다. 이미지 세트 탭은 기본적으로 선택됩니다.
3. 관련된 모든 가져오기 작업을 나열하려면 가져오기 탭을 선택합니다.

AWS CLI 및 SDKs

CLI

AWS CLI

dicom 가져오기 작업 나열

다음 `list-dicom-import-jobs` 코드 예시에서는 dicom 가져오기 작업을 나열합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 가져오기 작업 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListDICOMImportJobs를 참조하세요.

Java

SDK for Java 2.x

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

        try {
```

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

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListDICOMImportJobs를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

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

  return jobSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListDICOMImportJobs를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def list_dicom_import_jobs(self, datastore_id):
        """
        List the DICOM import jobs.

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListDICOMImportJobs를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

AWS HealthImaging으로 이미지 세트 액세스하기

AWS HealthImaging에서 의료 영상 데이터에 액세스하려면 일반적으로 고유 키가 있는 이미지 세트를 검색하고 관련 메타데이터 및 이미지 프레임(픽셀 데이터)을 가져와야 합니다.

> **중요**
>
> HealthImaging은 가져오는 동안 DICOM 인스턴스 바이너리(`.dcm` 파일)를 처리하고 이를 이미지 세트로 변환합니다. HealthImaging 클라우드 네이티브 작업(APIs)을 사용하여 데이터 스토어 및 이미지 세트를 관리합니다. HealthImaging의 DICOMweb 서비스 표현을 사용하여 DICOMweb 응답을 반환합니다.

다음 주제에서는 AWS Management Console AWS CLI및 AWS SDKs에서 HealthImaging 클라우드 네이티브 작업을 사용하여 이미지 세트를 검색하고 관련 속성, 메타데이터 및 이미지 프레임을 가져오는 방법을 설명합니다.

주제

- 이미지 세트 이해
- 이미지 세트 검색
- 이미지 세트 속성 가져오기
- 이미지 세트 메타데이터 가져오기
- 이미지 세트 픽셀 데이터 가져오기

이미지 세트 이해

이미지 세트는 DICOM 시리즈와 유사하며 AWS HealthImaging의 기반 역할을 AWS 하는 입니다. DICOM 데이터를 HealthImaging으로 가져올 때 이미지 세트가 생성됩니다. 서비스는 연구, 시리즈 및 인스턴스의 DICOM 계층 구조에 따라 가져온 P10 데이터를 구성하려고 시도합니다.

이미지 세트가 다음과 같은 이유로 도입되었습니다.

- 유연한 API를 통해 다양한 의료 영상 워크플로우 (임상 및 비임상)를 지원하십시오.
- 중복 및 일관되지 않은 데이터를 안정적으로 저장하고 조정할 수 있는 메커니즘을 제공합니다. 가져온 P10 데이터가 저장소에 이미 있는 기본 이미지 세트와 충돌하면 비기본으로 유지됩니다. 메타데이터 충돌을 해결한 후 데이터를 기본으로 만들 수 있습니다.

- 관련 데이터만 그룹화하여 환자 안전을 극대화.
- 데이터를 정리를 권장하여 불일치의 가시성 높이기. 자세한 내용은 이미지 세트 수정하기 단원을 참조하십시오.

> **중요**
>
> DICOM 데이터를 정리하기 전에 임상적으로 사용하면 환자에게 해를 끼칠 수 있습니다.

다음 메뉴는 이미지 세트를 더 자세히 설명하고 HealthImaging에서 이미지 세트의 기능과 용도를 이해하는 데 도움이 되는 예제와 다이어그램을 제공합니다.

이미지 세트란 무엇입니까?

이미지 세트는 DICOM 시리즈와 매우 유사한 관련 의료 영상 데이터를 최적화하기 위한 추상 그룹화 메커니즘을 정의하는 AWS 개념입니다. DICOM P10 영상 데이터를 AWS HealthImaging 데이터 스토어로 가져오면 메타데이터와 이미지 프레임(픽셀 데이터)으로 구성된 이미지 세트로 변환하는 것입니다.

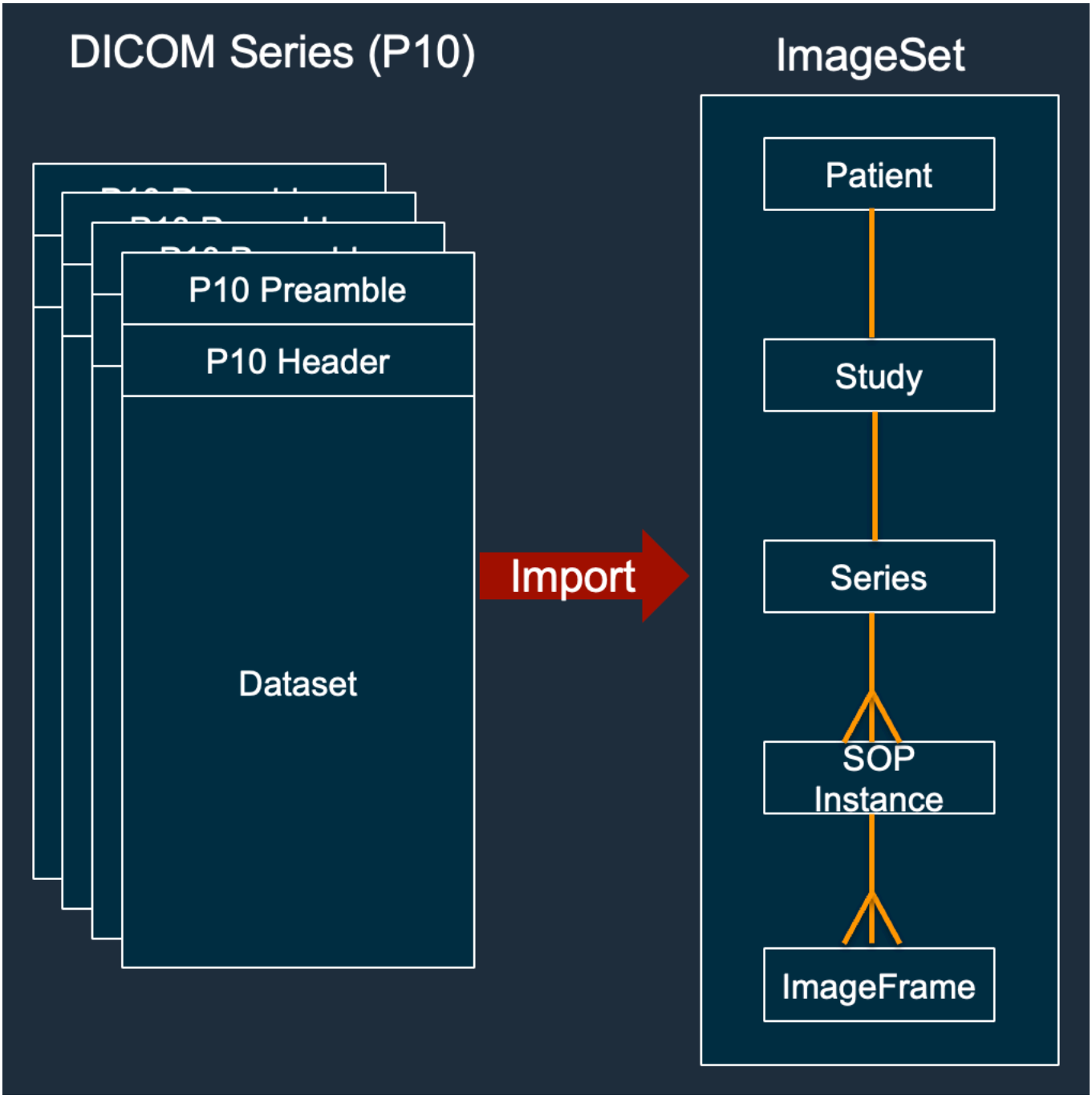
> **Note**
>
> 이미지 세트 메타데이터는 정규화됩니다. 즉, 하나의 공통 속성 및 값 세트가 Registry of DICOM Data Elements에 나열된 Patient, Study 및 Series 수준 요소에 매핑됩니다. HealthImaging은 수신 DICOM P10 객체를 이미지 세트로 그룹화할 때 다음 DICOM 요소를 사용합니다.
>
> 이미지 세트 생성에 사용되는 DICOM 요소
>
> | 요소 이름 | 요소 태그 |
> | --- | --- |
> | 연구 수준 요소 | |
> | `Study Date` | `(0008,0020)` |
> | `Accession Number` | `(0008,0050)` |
> | `Patient ID` | `(0010,0020)` |
> | `Study Instance UID` | `(0020,000D)` |

요소 이름	요소 태그
Study ID	(0020,0010)
시리즈 수준 요소	
Series Instance UID	(0020,000E)
Series Number	(0020,0011)

가져오는 동안 일부 이미지 세트는 원래 전송 구문 인코딩을 유지하는 반면, 다른 이미지 세트는 기본적으로 처리량이 높은 JPEG 2000(HTJ2K) 무손실로 트랜스코딩됩니다. 이미지 세트가 HTJ2K로 인코딩된 경우 보기 전에 디코딩해야 합니다. 자세한 내용은 지원되는 전송 구문 및 HTJ2K 디코딩 라이브러리 섹션을 참조하세요.
이미지 프레임(픽셀 데이터)은 HTJ2K(High-Throughput JPEG 2000)로 인코딩되며, 보기 전에 디코딩해야 합니다.

이미지 세트는 AWS 리소스이므로 Amazon 리소스 이름(ARNs)이 할당됩니다. 최대 50개의 키-값 페어로 태그를 지정할 수 있으며 IAM을 통해 역할 기반 액세스 제어(RBAC)와 ABAC(속성 기반 액세스 제어)를 부여받을 수 있습니다. 또한 이미지 세트에는 버전이 지정되므로 모든 변경 사항이 보존되고 이전 버전에 액세스할 수 있습니다.

DICOM P10 데이터를 가져오면 동일한 DICOM 시리즈에 있는 하나 이상의 서비스-객체 페어(SOP) 인스턴스에 대한 DICOM 메타데이터와 이미지 프레임이 포함된 이미지 세트가 생성됩니다.



> **Note**
>
> DICOM 가져오기 작업:
>
> - 항상 새 이미지 세트를 생성하거나 기존 이미지 세트의 버전을 증가시킵니다.

- SOP 인스턴스 스토리지를 중복 제거하지 마십시오. 동일한 SOP 인스턴스를 가져올 때마다 기본이 아닌 새 이미지 세트 또는 기존 기본 이미지 세트의 증분 버전으로 추가 스토리지를 사용합니다.
- 일관된 환자, 연구 및 시리즈 메타데이터 요소가 있는 인스턴스를 포함하는 일관되고 충돌하지 않는 메타데이터를 기본 이미지 세트로 사용하여 SOP 인스턴스를 자동으로 구성합니다.
  - DICOM 시리즈를 구성하는 인스턴스를 둘 이상의 가져오기 작업으로 가져오고 인스턴스가 데이터 스토어에 이미 있는 인스턴스와 충돌하지 않는 경우 모든 인스턴스가 하나의 기본 이미지 세트로 구성됩니다.
- 데이터 스토어에 이미 있는 기본 이미지 세트와 충돌하는 DICOM P10 데이터가 포함된 비기본 이미지 세트를 생성합니다.
- 가장 최근에 수신한 데이터를 기본 이미지 세트의 최신 버전으로 유지합니다.
  - DICOM 시리즈로 구성된 인스턴스가 기본 이미지 세트이고 인스턴스 하나를 다시 가져오면 새 복사본이 기본 이미지 세트에 삽입되고 버전이 증가합니다.

이미지 세트 메타데이터는 어떤 모습인가요?

`GetImageSetMetadata` 작업을 사용하여 이미지 세트 메타데이터를 검색합니다. 반환된 메타데이터는 로 압축`gzip`되므로 보기 전에 압축을 풀어야 합니다. 자세한 내용은 이미지 세트 메타데이터 가져오기 단원을 참조하십시오.

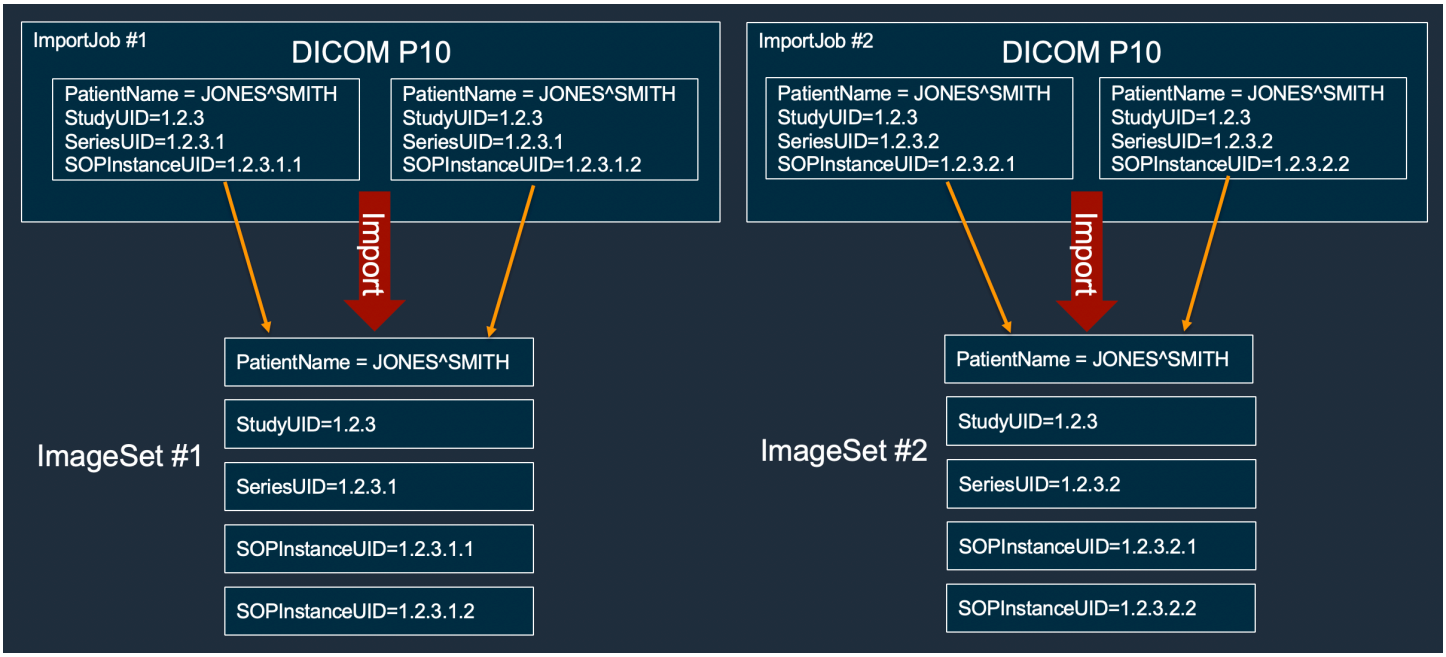
다음 예제에서는 이미지 세트 메타데이터의 구조를 JSON 형식으로 보여줍니다.

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

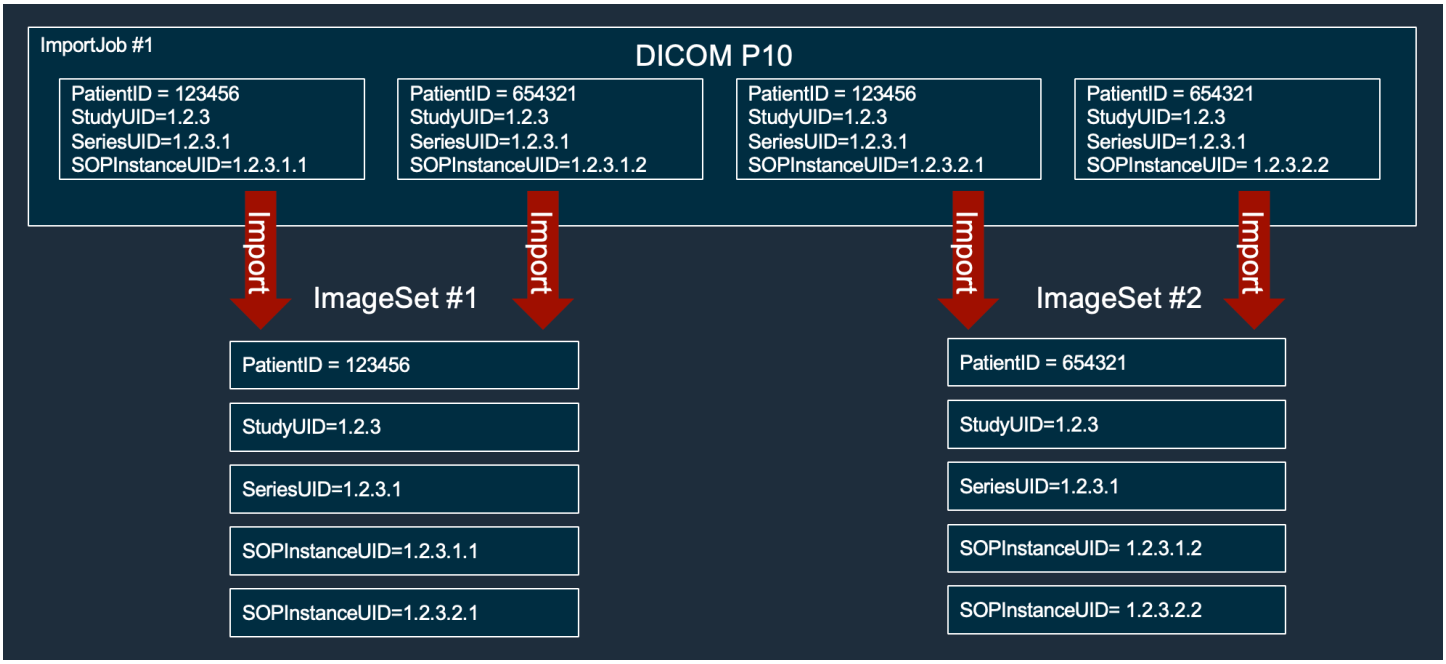
이미지 세트 생성 예제: 복수 가져오기 작업

다음 예제는 복수 가져오기 작업이 항상 새 이미지 세트를 생성하고 기존 이미지 세트에 절대로 추가하지 않는 방법을 보여줍니다.



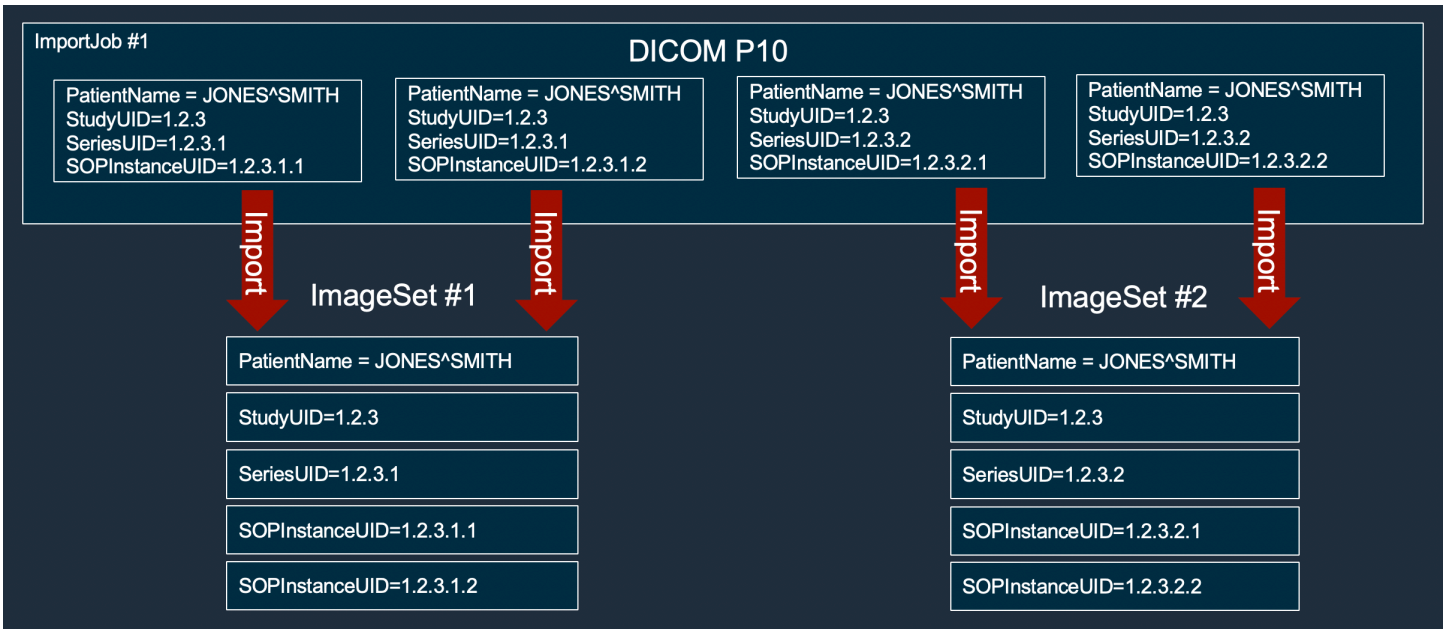
이미지 세트 생성 예: 두 가지 변형이 있는 단일 가져오기 작업

다음 예제에서는 인스턴스 1과 3의 환자 IDs가 인스턴스 2 및 4와 다르기 때문에 단일 이미지 세트로 병합되지 않는 단일 가져오기 작업을 보여줍니다. 이 문제를 해결하려면 `UpdateImageSetMetadata` 작업을 사용하여 기존 기본 이미지 세트와 환자 ID 충돌을 해결할 수 있습니다. 충돌이 해결되면 `CopyImageSet` 작업을 인수와 함께 사용하여 이미지 세트를 기본 이미지 세트에 `--promoteToPrimary` 추가할 수 있습니다.



이미지 세트 생성 예제: 최적화가 적용된 단일 가져오기 작업

다음 예제는 환자 이름이 일치하더라도 처리량을 늘리기 위해 두 개의 이미지 세트를 만드는 단일 가져오기 작업을 보여줍니다.



이미지 세트 검색

`SearchImageSets` 작업을 사용하여 `ACTIVE` HealthImaging 데이터 스토어의 모든 이미지 세트에 대해 검색 쿼리를 실행합니다. 다음 메뉴에서는 AWS Management Console 및 AWS SDKs에 대한 AWS

CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `SearchImageSets` 섹션을 참조하세요.

> **Note**
>
> 이미지 세트를 검색할 때 다음 사항에 유의하세요.
>
> - `SearchImageSets`는 단일 검색 쿼리 파라미터를 허용하고 일치하는 조건을 가진 모든 이미지 집합의 페이지 단위 응답을 반환합니다. 모든 날짜 범위 쿼리는 로 입력해야 합니다(`lowerBound, upperBound`).
> - 기본적으로는 `updatedAt` 필드를 `SearchImageSets` 사용하여 최신에서 오래된 순서로 정렬합니다.
> - 고객 소유 AWS KMS 키로 데이터 스토어를 생성한 경우 이미지 세트와 상호 작용하기 전에 AWS KMS 키 정책을 업데이트해야 합니다. 자세한 내용은 고객 관리형 키 생성을 참조하세요.

이미지 세트를 검색하려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

> **Note**
>
> 다음 절차에서는 `Series Instance UID` 및 `Updated at` 속성 필터를 사용하여 이미지 세트를 검색하는 방법을 보여줍니다.

Series Instance UID

`Series Instance UID` 속성 필터를 사용하여 이미지 세트 검색

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열리고 기본적으로 이미지 세트 탭이 선택됩니다.
3. 속성 필터 메뉴를 선택하고를 선택합니다`Series Instance UID`.
4. 검색할 값 입력 필드에 관심 있는 시리즈 인스턴스 UID를 입력(붙여넣기)합니다.

> **Note**
>
> 시리즈 인스턴스 UID 값은 DICOM 고유 식별자(UIDs) 레지스트리에 나열된 값과 동일해야 합니다. 요구 사항에는 둘 사이에 기간이 하나 이상 포함된 일련의 숫자가 포함됩니다. 기간은 시리즈 인스턴스 UIDs. 문자와 공백은 허용되지 않으므로 UIDs.

5. 날짜 범위 메뉴를 선택하고 시리즈 인스턴스 UID의 날짜 범위를 선택한 다음 적용을 선택합니다.
6. 검색을 선택합니다.

   선택한 날짜 범위에 속하는 시리즈 인스턴스 UIDs는 기본적으로 최신 순서로 반환됩니다.

Updated at

`Updated at` 속성 필터를 사용하여 이미지 세트 검색

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열리고 기본적으로 이미지 세트 탭이 선택됩니다.
3. 속성 필터 메뉴를 선택하고를 선택합니다`Updated at`.
4. 날짜 범위 메뉴를 선택하고 이미지 세트 날짜 범위를 선택한 다음 적용을 선택합니다.
5. 검색을 선택합니다.

   선택한 날짜 범위에 속하는 이미지 세트는 기본적으로 최신 순서로 반환됩니다.

AWS CLI 및 SDKs

C++

SDK for C++

이미지 세트 검색을 위한 유틸리티 함수.

```
//! 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;
}
```

사용 사례 #1: EQUAL 연산자.

```
        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;
            }
        }
```

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

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

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

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

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

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

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

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

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

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

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
       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;
           }
       }
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
       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;
            }
        }
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 SearchImageSets를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

예시 1: EQUAL 연산자로 이미지 세트 검색

다음 `search-image-sets` 코드 예시에서는 EQUAL 연산자를 사용하여 특정 값을 기준으로 이미지 세트를 검색합니다.

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

`search-criteria.json`의 콘텐츠

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

출력:

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

예시 2: DICOMStudyDate 및 DICOMStudyTime을 사용하여 BETWEEN 연산자로 이미지 세트 검색

다음 `search-image-sets` 코드 예시에서는 1990년 1월 1일(오전 12시)에서 2023년 1월 1일(오전 12시) 사이에 생성된 DICOM 연구가 있는 이미지 세트를 검색합니다.

참고: DICOMStudyTime은 선택 사항입니다. 해당 날짜가 없는 경우 필터링에 제공되는 날짜의 시간 값은 오전 12시(하루의 시작)입니다.

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

`search-criteria.json`의 콘텐츠

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

출력:

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

예시 3: CreatedAt을 사용하여 BETWEEN 연산자로 이미지 세트 검색(시간 연구가 이전에 지속됨)

다음 `search-image-sets` 코드 예시에서는 UTC 시간대의 시간 범위에서 HealthImaging에 대한 DICOM 연구가 지속된 이미지 세트를 검색합니다.

참고: createdAt을 예시 형식('1985-04-12T23:20:50.52Z')으로 제공합니다.

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

`search-criteria.json`의 콘텐츠

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

```
        "operator": "BETWEEN"
    }]
}
```

출력:

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

예시 4: updatedAt의 DICOMSeriesInstanceUID 및 BETWEEN에서 EQUAL 연산자로 이미지 세트를 검색하고 updatedAt 필드의 ASC 순서로 응답 정렬

다음 `search-image-sets` 코드 예시에서는 DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN을 사용하여 이미지 세트를 검색하고 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

참고: updatedAt을 예시 형식('1985-04-12T23:20:50.52Z')으로 제공합니다.

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

`search-criteria.json`의 콘텐츠

```
{
    "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"
    }
}
```

출력:

```
{
    "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"
    }]
```

```
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 검색을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 SearchImageSets를 참조하세요.

Java

SDK for Java 2.x

이미지 세트 검색을 위한 유틸리티 함수.

```
    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;
    }
```

사용 사례 #1: EQUAL 연산자.

```
        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();
        }
```

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

```
        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();
        }
```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
        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();
        }
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
        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();
        }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 SearchImageSets를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

이미지 세트 검색을 위한 유틸리티 함수.

```
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;
};
```

사용 사례 #1: EQUAL 연산자.

```
  const datastoreId = "12345678901234567890123456789012";

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

    await searchImageSets(datastoreId, searchCriteria);
```

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

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

```
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);
}
```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
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);
  }
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
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);
  }
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 SearchImageSets를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

이미지 세트 검색을 위한 유틸리티 함수.

```
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
```

사용 사례 #1: EQUAL 연산자.

```
        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}")
```

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

```
        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}"
        )
```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
        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}"
        )
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
        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}")
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 SearchImageSets를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

이미지 세트 속성 가져오기

`GetImageSet` 작업을 사용하여 HealthImaging에서 지정된 이미지 세트의 속성을 반환합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `GetImageSet` 섹션을 참조하세요.

> **Note**
>
> 기본적으로 AWS HealthImaging은 최신 버전의 이미지 세트에 대한 속성을 반환합니다. 이전 버전의 이미지 세트에 대한 속성을 보려면 요청과 함께 `versionId`를 제공하십시오. `GetDICOMInstanceHealthImaging`의 DICOMweb 서비스 표현인를 사용하여 DICOM 인스턴스 바이너리(`.dcm` 파일)를 반환합니다. 자세한 내용은 HealthImaging에서 DICOM 인스턴스 가져오기 단원을 참조하십시오.

이미지 세트 속성 가져오기

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열리고 기본적으로 이미지 세트 탭이 선택됩니다.
3. 이미지 세트를 선택합니다.

   이미지 세트 세부 정보 페이지가 열리고 이미지 세트 속성이 표시됩니다.

AWS CLI 및 SDKs

CLI

AWS CLI

이미지 세트 속성 가져오기

다음 `get-image-set` 코드 예시에서는 이미지 세트의 속성을 가져옵니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 속성 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetImageSet를 참조하세요.

Java

SDK for Java 2.x

```
   public static GetImageSetResponse getMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imagesetId,
           String versionId) {
       try {
           GetImageSetRequest.Builder getImageSetRequestBuilder =
GetImageSetRequest.builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId);

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

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

       return null;
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

```
  // }

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.health_imaging_client.get_image_set(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set = self.health_imaging_client.get_image_set(
```

```
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
        except ClientError as err:
            logger.error(
                "Couldn't get image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return image_set
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetImageSet를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

이미지 세트 메타데이터 가져오기

`GetImageSetMetadata` 작업을 사용하여 HealthImaging에서 지정된 이미지 세트의 메타데이터를 검색합니다. 다음 메뉴는 및 AWS SDKs에 AWS Management Console 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `GetImageSetMetadata` 섹션을 참조하세요.

> **Note**
>
> 기본적으로 HealthImaging은 최신 버전의 이미지 세트에 대한 메타데이터 속성을 반환합니다. 이전 버전의 이미지 세트에 대한 메타데이터를 보려면 요청과 함께 `versionId`를 제공하십시오.
>
> 이미지 세트 메타데이터는 `gzip`으로 압축되어 JSON 객체로 반환됩니다. 따라서 정규화된 메타데이터를 보기 전에 JSON 객체의 압축을 풀어야 합니다. 자세한 내용은 메타데이터 정규화 단원을 참조하십시오.
>
> `GetDICOMInstanceMetadata`HealthImaging의 DICOMweb 서비스 표현인를 사용하여 DICOM 인스턴스 메타데이터(`.json` 파일)를 반환합니다. 자세한 내용은 HealthImaging에서 DICOM 인스턴스 메타데이터 가져오기 단원을 참조하십시오.

이미지 세트 메타데이터를 가져오려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열리고 기본적으로 이미지 세트 탭이 선택됩니다.
3. 이미지 세트를 선택합니다.

   이미지 세트 세부 정보 페이지가 열리고 이미지 세트 메타데이터가 이미지 세트 메타데이터 뷰어 섹션 아래에 표시됩니다.

AWS CLI 및 SDKs

C++

SDK for C++

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

```
//! Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
```

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

    return outcome.IsSuccess();
}
```

버전 없이 이미지 세트 메타데이터를 가져옵니다.

```
        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;
        }
```

버전과 함께 이미지 세트 메타데이터를 가져옵니다.

```
        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 세부 정보는 AWS SDK for C++ API 참조의 GetImageSetMetadata를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

예시 1: 버전 없이 이미지 세트 메타데이터 가져오기

다음 `get-image-set-metadata` 코드 예시에서는 버전을 지정하지 않고 이미지 세트의 메타데이터를 가져옵니다.

참고: `outfile`은 필수 파라미터입니다.

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

반환된 메타데이터는 gzip으로 압축되어 studymetadata.json.gz 파일에 저장됩니다. 반환된 JSON 객체의 콘텐츠를 보려면 먼저 압축을 풀어야 합니다.

출력:

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

예시 2: 버전과 함께 이미지 세트 메타데이터 가져오기

다음 `get-image-set-metadata` 코드 예시에서는 지정된 버전의 이미지 세트에 대한 메타데이터를 가져옵니다.

참고: `outfile`은 필수 파라미터입니다.

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

반환된 메타데이터는 gzip으로 압축되어 studymetadata.json.gz 파일에 저장됩니다. 반환된 JSON 객체의 콘텐츠를 보려면 먼저 압축을 풀어야 합니다.

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 메타데이터 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetImageSetMetadata를 참조하세요.

Java

SDK for Java 2.x

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

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

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


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

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetImageSetMetadata를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

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

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

버전 없이 이미지 세트 메타데이터를 가져옵니다.

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

버전과 함께 이미지 세트 메타데이터를 가져옵니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetImageSetMetadata를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

```
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
```

버전 없이 이미지 세트 메타데이터를 가져옵니다.

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

버전과 함께 이미지 세트 메타데이터를 가져옵니다.

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetImageSetMetadata를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

이미지 세트 픽셀 데이터 가져오기

이미지 프레임은 2D 의료 영상을 구성하기 위해 이미지 세트 내에 존재하는 픽셀 데이터입니다. `GetImageFrame` 작업을 사용하여 HealthImaging에서 지정된 이미지 세트에 대한 HTJ2K-encoded 이미지 프레임을 검색합니다. ??? 다음 메뉴에서는 AWS CLI 및 AWS SDKs에 대한 코드 예제를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `GetImageFrame` 섹션을 참조하세요.

> **Note**
>
> `GetImageFrame` 작업을 사용할 때는 다음 사항에 유의하세요.
>
> - 가져오는 동안 HealthImaging은 일부 전송 구문에 대한 인코딩을 유지하지만 기본적으로 다른를 HTJ2K 무손실로 트랜스코딩합니다. 따라서 이미지 뷰어에서 보기 전에 이미지 프레임을 디코딩해야 합니다. 자세한 내용은 지원되는 전송 구문 및 HTJ2K 디코딩 라이브러리 섹션을 참조하세요.
> - 전송 구문의 MPEG 패밀리(MPEG2, MPEG-4 AVC/H.264 및 HEVC/H.265 포함)에 인코딩된 하나 이상의 이미지 프레임을 사용하여 HealthImaging에 저장된 인스턴스의 경우 `GetImageFrame` 작업은 저장된 전송 구문에서 비디오 객체를 반환합니다.
> - `GetImageFrame` 작업은 기본적으로 인스턴스의 저장된 전송 구문으로 이미지 프레임을 반환합니다. 자세한 내용은 지원되는 전송 구문 단원을 참조하십시오.
> - `GetDICOMInstanceFrames`또한 HealthImaging의 DICOMweb 서비스 표현인를 사용하여 DICOMweb 호환 뷰어 및 애플리케이션에 대한 DICOM 인스턴스 프레임(`multipart` 요청)

을 검색할 수 있습니다. 자세한 내용은 HealthImaging에서 DICOM 인스턴스 프레임 가져오기 단원을 참조하십시오.

이미지 세트 픽셀 데이터 가져오기

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

> **Note**
>
> AWS Management Console에서는 이미지 뷰어를 사용할 수 없으므로 이미지 프레임은 프로그래밍 방식으로 액세스하고 디코딩해야 합니다.
> 이미지 프레임 디코딩 및 보기에 대한 자세한 내용은 HTJ2K 디코딩 라이브러리 섹션을 참조하십시오.

AWS CLI 및 SDKs

C++

SDK for C++

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

    Aws::MedicalImaging::Model::GetImageFrameRequest request;
```

```
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);

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

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

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

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

    }

    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 GetImageFrame을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

이미지 세트 픽셀 데이터 가져오기

다음 `get-image-frame` 코드 예시에서는 이미지 프레임을 가져옵니다.

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

참고: GetImageFrame 작업이 픽셀 데이터 스트림을 imageframe.jph 파일에 반환하므로 이 코드 예시에는 출력이 포함되지 않습니다. 이미지 프레임 디코딩 및 보기에 대한 자세한 내용은 HTJ2K 디코딩 라이브러리를 참조하세요.

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 픽셀 데이터 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetImageFrame을 참조하세요.

Java

SDK for Java 2.x

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

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

.imageFrameInformation(ImageFrameInformation.builder()

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

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,
```

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

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetImageFrame을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
```

```
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetImageFrame을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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

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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetImageFrame을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

AWS HealthImaging으로 이미지 세트 수정하기

DICOM 가져오기 작업에서는 일반적으로 다음과 같은 이유로 이미지 세트를 수정해야 합니다.

- 환자 안전
- 데이터 일관성
- 스토리지 비용 감소

> **중요**
>
> HealthImaging은 가져오는 동안 DICOM 인스턴스 바이너리(`.dcm` 파일)를 처리하고 이를 이미지 세트로 변환합니다. HealthImaging 클라우드 네이티브 작업(APIs)을 사용하여 데이터 스토어 및 이미지 세트를 관리합니다. HealthImaging의 DICOMweb 서비스 표현을 사용하여 DICOMweb 응답을 반환합니다.

HealthImaging은 여러 클라우드 네이티브 APIs를 제공하여 이미지 세트 수정 프로세스를 간소화합니다. 다음 주제에서는 AWS CLI 및 AWS SDKs를 사용하여 이미지 세트를 수정하는 방법을 설명합니다.

주제

- 이미지 세트 버전 목록
- 이미지 세트 메타데이터 업데이트
- 이미지 세트 복사
- 이미지 세트 삭제

이미지 세트 버전 목록

`ListImageSetVersions` 작업을 사용하여 HealthImaging에서 이미지 세트의 버전 기록을 나열합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `ListImageSetVersions` 섹션을 참조하세요.

> **Note**
>
> AWS HealthImaging은 이미지 세트의 모든 변경 사항을 기록합니다. 이미지 세트 메타데이터를 업데이트하면 이미지 세트 기록에 새 버전이 생성됩니다. 자세한 내용은 이미지 세트 메타데이터 업데이트 단원을 참조하십시오.

이미지 세트의 버전을 나열하려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열리고 기본적으로 이미지 세트 탭이 선택됩니다.
3. 이미지 세트를 선택합니다.

   이미지 세트 세부정보 페이지가 열립니다.

   이미지 세트 버전은 이미지 세트 세부정보 섹션 아래에 표시됩니다.

AWS CLI 및 SDKs

CLI

AWS CLI

이미지 세트 버전 나열

다음 `list-image-set-versions` 코드 예시에서는 이미지 세트의 버전 기록을 나열합니다.

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

출력:

```
{
```

```
    "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
        }
    ]
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 버전 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListImageSetVersions를 참조하세요.

Java

SDK for Java 2.x

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

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

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

       return null;
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListImageSetVersions를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

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

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListImageSetVersions를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :return: The list of image set versions.
        """
        try:
            paginator = self.health_imaging_client.get_paginator(
                "list_image_set_versions"
            )
            page_iterator = paginator.paginate(
```

```
                imageSetId=image_set_id, datastoreId=datastore_id
            )
            image_set_properties_list = []
            for page in page_iterator:
                image_set_properties_list.extend(page["imageSetPropertiesList"])
        except ClientError as err:
            logger.error(
                "Couldn't list image set versions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return image_set_properties_list
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListImageSetVersions를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

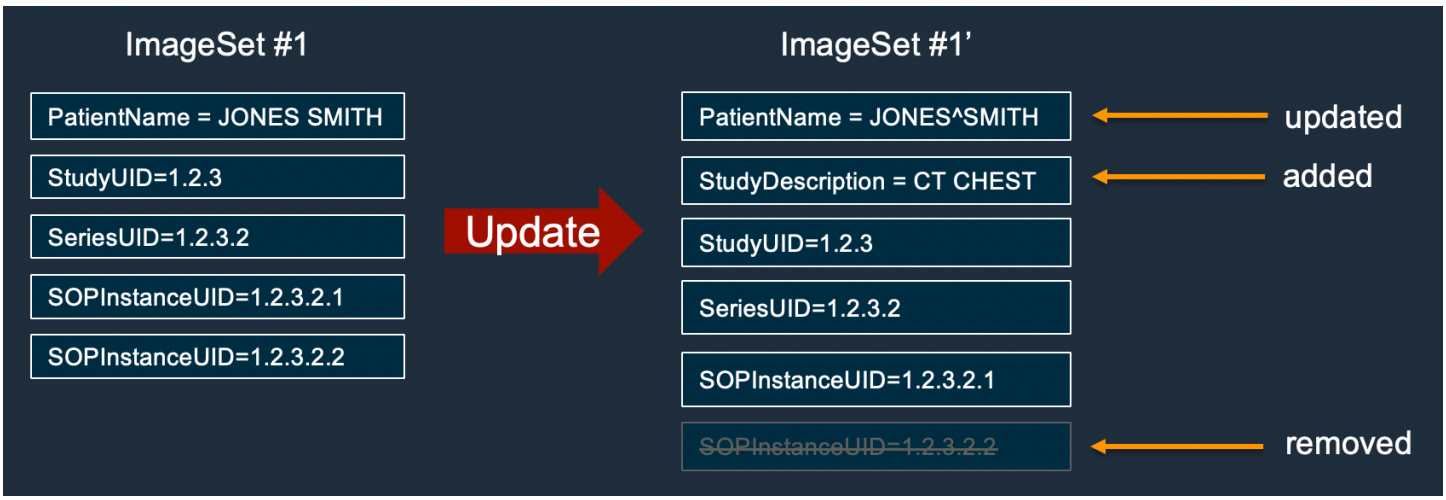
이미지 세트 메타데이터 업데이트

UpdateImageSetMetadata 작업을 사용하여 AWS HealthImaging에서 이미지 세트 메타데이터를 업데이트합니다. 이 비동기 프로세스를 사용하여 가져오기 중에 생성되는 DICOM 정규화 요소의 표현인 이미지 세트 메타데이터 속성을 추가, 업데이트 및 제거할 수 있습니다. 이 UpdateImageSetMetadata 작업을 사용하면 시리즈 및 SOP 인스턴스를 제거하여 이미지 세트를 외부 시스템과 동기화하고 이미지 세트 메타데이터를 식별할 수 없도록 할 수도 있습니다. 자세한 내용은 AWS HealthImaging API Reference에서 UpdateImageSetMetadata 섹션을 참조하세요.

> **Note**
>
> 실제 DICOM을 가져오려면 이미지 세트 메타데이터에서 속성을 업데이트, 추가 및 제거해야 합니다. 이미지 세트 메타데이터를 업데이트할 때는 다음 사항에 유의하세요.
>
> - 이미지 세트 메타데이터를 업데이트하면 이미지 세트 기록에 새 버전이 생성됩니다. 자세한 내용은 이미지 세트 버전 목록 단원을 참조하십시오. 이전 이미지 세트 버전 ID로 되돌리려면 선택적 revertToVersionId 파라미터를 사용합니다.
> - 이미지 세트 메타데이터 업데이트는 비동기식 프로세스입니다. 따라서 imageSetState 및 imageSetWorkflowStatus 응답 요소를 사용하여 업데이트 중인 이미지 세트의 각 상태와 상태를 제공할 수 있습니다. LOCKED 이미지 세트에 대해 다른 쓰기 작업을 수행할 수 없습니다.
> - UpdateImageSetMetadata 작업이 성공하지 못하면를 호출하고 message 응답 요소를 검토하여를 확인합니다common errors.
> - DICOM 요소 제약 조건은 메타데이터 업데이트에 적용됩니다. force 요청 파라미터를 사용하면를 재정의하려는 경우 기본이 아닌 이미지 세트의 요소를 업데이트할 수 있습니다DICOM 메타데이터 제약 조건.
> - 기본 이미지 세트의 경우 환자 및 시리즈 수준 메타데이터 요소를 업데이트할 수 없습니다. UpdateImageSet는 기본 이미지 세트에 대해 StudyInstanceUID, SeriesInstanceUID 및 SOPInstanceUID를 업데이트하는 --force를 지원하지 않습니다.
> - 기본이 아닌 이미지 세트에서 UpdateImageSetMetadata 작업을 강제로 완료하도록 force 요청 파라미터를 설정합니다. 이 파라미터를 설정하면 이미지 세트를 다음과 같이 업데이트할 수 있습니다.
> - Tag.StudyInstanceUID, Tag.SeriesInstanceUIDTag.SOPInstanceUID, 및 Tag.StudyID 속성 업데이트
> - 인스턴스 수준 프라이빗 DICOM 데이터 요소 추가, 제거 또는 업데이트

- 이미지 세트를 기본으로 승격하면 이미지 세트 ID가 변경됩니다.

다음 다이어그램은 HealthImaging에서 업데이트되는 이미지 세트 메타데이터를 나타냅니다.



이미지 세트 메타데이터 업데이트

AWS HealthImaging에 대한 액세스 기본 설정에 따라 탭을 선택합니다.

AWS CLI 및 SDKs

CLI

AWS CLI

예시 1: 이미지 세트 메타데이터에 속성을 삽입하거나 업데이트

다음 `update-image-set-metadata` 예시에서는 이미지 세트 메타데이터의 속성을 삽입하거나 업데이트합니다.

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json`의 콘텐츠

```
{
```

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

출력:

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

예시 2: 이미지 세트 메타데이터에서 속성 제거

다음 `update-image-set-metadata` 예시에서는 이미지 세트 메타데이터에서 속성을 제거합니다.

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json`의 콘텐츠

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

출력:

```
{
```

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

예시 3: 이미지 세트 메타데이터에서 인스턴스 제거

다음 `update-image-set-metadata` 예시에서는 이미지 세트 메타데이터에서 인스턴스를 제거합니다.

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates file://metadata-updates.json
```

`metadata-updates.json`의 콘텐츠

```
{
    "DICOMUpdates": {
        "removableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\":
 {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}"
    }
}
```

출력:

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

예시 4: 이미지 세트를 이전 버전으로 되돌리기

다음 `update-image-set-metadata` 예시에서는 이미지 세트를 이전 버전으로 되돌리는 방법을 보여줍니다. CopyImageSet 및 UpdateImageSetMetadata 작업은 새 버전의 이미지 세트를 생성합니다.

```
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"}'
```

출력:

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

예시 5: 인스턴스에 프라이빗 DICOM 데이터 요소 추가

다음 `update-image-set-metadata` 예시에서는 이미지 세트 내 지정된 인스턴스에 프라이빗 요소를 추가하는 방법을 보여줍니다. DICOM 표준은 표준 데이터 요소에 포함할 수 없는 정보를 통신하기 위한 프라이빗 데이터 요소를 허용합니다. UpdateImageSetMetadata 작업을 사용하여 프라이빗 데이터 요소를 생성, 업데이트 및 삭제할 수 있습니다.

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

`metadata-updates.json`의 콘텐츠

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":
 {\"001910F9\": \"97\"},\"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}"
    }
}
```

출력:

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

예시 6: 프라이빗 DICOM 데이터 요소를 인스턴스로 업데이트

다음 `update-image-set-metadata` 예시에서는 이미지 세트 내 인스턴스에 속하는 프라이빗 데이터 요소의 값을 업데이트하는 방법을 보여줍니다.

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

`metadata-updates.json`의 콘텐츠

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

```
    }
}
```

출력:

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

예시 7: SOPInstanceUID를 강제 파라미터로 업데이트

다음 `update-image-set-metadata` 예시에서는 강제 파라미터를 사용하여 DICOM 메타데이터 제약 조건을 재정의하여 SOPInstanceUID를 업데이트하는 방법을 보여줍니다.

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

`metadata-updates.json`의 콘텐츠

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 메타데이터 업데이트를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateImageSetMetadata를 참조하세요.

Java

SDK for Java 2.x

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

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

            UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

사용 사례 #1: 속성을 삽입하거나 업데이트합니다.

```
                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);
```

사용 사례 #2: 속성을 제거합니다.

```
                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);
```

사용 사례 #3: 인스턴스를 제거합니다.

```
                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);
```

사용 사례 #4: 이전 버전으로 되돌립니다.

```
                // 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);
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateImageSetMetadata를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
```

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

사용 사례 #1: 속성을 삽입 또는 업데이트하고 강제로 업데이트합니다.

```
    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,
    );
```

사용 사례 #2: 속성을 제거합니다.

```
    // 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,
  );
```

사용 사례 #3: 인스턴스를 제거합니다.

```
  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,
    );
```

사용 사례 #4: 이전 버전으로 되돌립니다.

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

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateImageSetMetadata를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def update_image_set_metadata(
```

```
        self, datastore_id, image_set_id, version_id, metadata, force=False
    ):
        """
        Update the metadata of an image set.

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

사용 사례 #1: 속성을 삽입하거나 업데이트합니다.

```
            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
            )
```

사용 사례 #2: 속성을 제거합니다.

```
            # 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
            )
```

사용 사례 #3: 인스턴스를 제거합니다.

```
            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
            )
```

사용 사례 #4: 이전 버전으로 되돌립니다.

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

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateImageSetMetadata을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

`CopyImageSet`, `UpdateImageSetMetadata`및 `DeleteImageSet` APIs.

`DeleteImageSet` 작업을 사용하여 기본 컬렉션에서 이미지 세트를 제거할 수 있습니다.

기본 이미지 세트의 메타데이터를 업데이트하려면

1. CopyImageSet 작업을 사용하여 수정하려는 기본 이미지 세트의 복사본인 비기본 이미지 세트를 생성합니다. 가 기본이 아닌 이미지 세트 ID`103785414bc2c89330f7ce51bbd13f7a`로 반환한다고 가정해 보겠습니다.

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

2. UpdateImageSetMetadata 작업을 사용하여 기본이 아닌 이미지 세트를 변경합니다(`103785414bc2c89330f7ce51bbd13f7a`). 예를 들어, PatientID를 변경합니다.

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

3. 수정하려는 기본 이미지 세트를 삭제합니다.

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

4. CopyImageSet 작업을 인수와 함께 사용하여 업데이트된 이미지 세트를 기본 컬렉션에 `--promoteToPrimary` 추가합니다.

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

5. 기본이 아닌 이미지 세트를 삭제합니다.

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

기본이 아닌 이미지 세트를 기본으로 만들려면

1. UpdateImageSetMetadata 작업을 사용하여 기존 기본 이미지 세트와의 충돌을 해결합니다.

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

2. 충돌이 해결되면 CopyImageSet 작업을 인수와 함께 사용하여 이미지 세트를 기본 이미지 세트 컬렉션에 `--promoteToPrimary` 추가합니다.

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

3. CopyImageSet 작업이 성공했는지 확인한 후 소스 비기본 이미지 세트를 삭제합니다.

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

이미지 세트 복사

CopyImageSet 작업을 사용하여 HealthImaging에서 이미지 세트를 복사합니다. 사용자는 이 비동기 프로세스를 사용하여 이미지 세트의 내용을 새 이미지 세트 또는 기존 이미지 세트로 복사합니다. 새 이미지 세트로 복사하여 이미지 세트를 분할하고 별도의 복사본을 생성할 수 있습니다. 기존 이미지 세트에 복사하여 두 이미지 세트를 병합할 수도 있습니다. 자세한 내용은 AWS HealthImaging API Reference에서 CopyImageSet 섹션을 참조하세요.

> **Note**
>
> CopyImageSet 작업을 사용할 때는 다음 사항에 유의하세요.
>
> - CopyImageSet 작업은 새 이미지 세트 또는 새 버전의를 생성합니다destinationImageSet. 자세한 내용은 이미지 세트 버전 목록 단원을 참조하십시오.
> - 복사는 비동기 프로세스입니다. 따라서 상태(imageSetState) 및 상태(imageSetWorkflowStatus) 응답 요소를 사용하여 잠긴 이미지 세트에서 어떤 작업이 발생하고 있는지 알 수 있습니다. 잠긴 이미지 세트에서는 다른 쓰기 작업을 수행할 수 없습니다.
> - CopyImageSet 에서는 SOP 인스턴스 UIDs 이미지 세트 내에서 고유해야 합니다.
> - 를 사용하여 SOP 인스턴스의 하위 집합을 복사할 수 있습니다copiableAttributes. 이렇게 하면에서 로 복사sourceImageSet할 SOP 인스턴스를 하나 이상 선택할 수 있습니다destinationImageSet.
> - CopyImageSet 작업이 성공하지 못하면를 호출GetImageSet하고 message 속성을 검토합니다. 자세한 내용은 이미지 세트 속성 가져오기 단원을 참조하십시오.
> - 실제 DICOM을 가져오면 DICOM 시리즈당 여러 이미지 세트가 생성될 수 있습니다. 선택적 force 파라미터가 제공되지 않는 한 CopyImageSet 작업에는 sourceImageSet 및 destinationImageSet에 일관된 메타데이터가 있어야 합니다.
> - sourceImageSet와 사이에 일관되지 않은 메타데이터 요소가 있더라도 작업을 강제로 수행하도록 force 파라미터를 설정합니다destinationImageSet. 이러한 경우 환자, 연구 및 시리즈 메타데이터는에서 변경되지 않습니다destinationImageSet.

이미지 세트를 복사하려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 탭을 선택합니다.

AWS CLI 및 SDKs

CLI

AWS CLI

예시 1: 대상 없이 이미지 세트 복사

다음 `copy-image-set` 예시에서는 대상 없이 이미지 세트의 복제본을 만듭니다.

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

출력:

```
{
    "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"
}
```

예시 2: 대상과 함께 이미지 세트 복사

다음 copy-image-set 예시에서는 대상과 함께 이미지 세트의 복제본을 만듭니다.

```
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"} }'
```

출력:

```
{
    "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"
}
```

예시 3: 소스 이미지 세트의 인스턴스 하위 집합을 대상 이미지 세트로 복사

다음 copy-image-set 예시에서는 소스 이미지 세트의 DICOM 인스턴스 하나를 대상 이미지 세트로 복사합니다. 강제 파라미터는 환자, 연구 및 시리즈 수준 속성의 불일치를 재정의하기 위해 제공됩니다.

```
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
```

출력:

```
{
    "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"
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 복사를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CopyImageSet를 참조하세요.

Java

SDK for Java 2.x

```
    /**
     * Copy an AWS HealthImaging image set.
     *
     * @param medicalImagingClient  - The AWS HealthImaging client object.
```

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

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

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

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

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

           // Optionally designate a destination image set.
```

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

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

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

복사 가능한 속성을 생성하는 유틸리티 함수입니다.

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 CopyImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

이미지 세트를 복사하는 유틸리티 함수입니다.

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination
 image set.
 * @param {string} destinationVersionId - The optional version ID of the
 destination image set.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "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: 'xxxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
```

```
    //             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);
  }
};
```

대상 없이 이미지 세트를 복사합니다.

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

대상이 있는 이미지 세트를 복사합니다.

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

이미지 세트의 하위 집합을 대상으로 복사하고 강제로 복사합니다.

```
  await copyImageSet(
```

```
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
    "12345678901234567890123456789012",
    "1",
    true,
    ["12345678901234567890123456789012", "11223344556677889900112233445566"],
  );
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 CopyImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

이미지 세트를 복사하는 유틸리티 함수입니다.

```
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"]
```

대상 없이 이미지 세트를 복사합니다.

```
            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,
            )
```

대상이 있는 이미지 세트를 복사합니다.

```
            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,
            )
```

이미지 세트의 하위 집합을 복사합니다.

```
            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,
            )
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CopyImageSet를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

이미지 세트 삭제

`DeleteImageSet` 작업을 사용하여 HealthImaging에서 이미지 세트를 삭제합니다. 다음 메뉴는 AWS Management Console 및 AWS SDKs에 대한 AWS CLI 및 코드 예제에 대한 절차를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `DeleteImageSet` 섹션을 참조하세요.

이미지 세트를 삭제하려면

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열리고 기본적으로 이미지 세트 탭이 선택됩니다.
3. 이미지 세트를 선택하고 삭제를 선택합니다.

   이미지 세트 삭제 모달이 열립니다.
4. 이미지 세트의 ID를 제공하고 이미지 세트 삭제를 선택합니다.

AWS CLI 및 SDKs

C++

SDK for C++

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

    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 DeleteImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

이미지 세트 삭제

다음 `delete-image-set` 코드 예시에서는 이미지 세트를 삭제합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteImageSet를 참조하세요.

Java

SDK for Java 2.x

```
    public static void deleteMedicalImageSet(MedicalImagingClient
 medicalImagingClient,
            String datastoreId,
            String imagesetId) {
        try {
```

```
            DeleteImageSetRequest deleteImageSetRequest =
 DeleteImageSetRequest.builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId)
                    .build();

            medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
```

```
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //    '$metadata': {
  //        httpStatusCode: 200,
  //        requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //        extendedRequestId: undefined,
  //        cfId: undefined,
  //        attempts: 1,
  //        totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def delete_image_set(self, datastore_id, image_set_id):
        """
        Delete an image set.
```

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteImageSet를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

AWS HealthImaging으로 리소스에 태그 지정

태그 형태로 HealthImaging 리소스(데이터 스토어 및 이미지 세트)에 메타데이터를 할당할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다. 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색 및 필터링할 수 있습니다.

> **중요**
>
> 개인 식별 정보(PII)나 기타 기밀 정보 또는 민감한 정보를 태그에 저장하지 마십시오. 태그는 개인 데이터나 민감한 데이터에 사용하기 위한 것이 아닙니다.

다음 주제에서는 AWS Management Console AWS CLI, 및 AWS SDKs를 사용하여 HealthImaging 태그 지정 작업을 사용하는 방법을 설명합니다. 자세한 내용은 AWS 일반 참조 가이드의 AWS 리소스 태그 지정을 참조하세요.

주제

- 리소스에 태그 지정
- 리소스에 대한 태그 나열
- 리소스의 태그 해제

리소스에 태그 지정

`TagResource` 작업을 사용하여 AWS HealthImaging의 데이터 스토어 및 이미지 세트에 태그를 지정합니다. 다음 코드 예제에서는 AWS Management Console AWS CLI, 및 AWS SDKs에서 `TagResource` 작업을 사용하는 방법을 설명합니다. 자세한 내용은 AWS 일반 참조 가이드의 AWS 리소스 태그 지정을 참조하세요.

리소스에 태그 지정

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

데이터 스토어 세부 정보 페이지가 열립니다.

3. 세부 정보 탭을 선택하십시오.
4. 태그 섹션에서 태그 관리를 선택합니다.

   태그 관리 페이지가 열립니다.

5. 새 태그 추가를 선택합니다.
6. 키와 값(선택 사항)을 입력합니다.
7. Save changes(변경 사항 저장)를 선택합니다.

AWS CLI 및 SDKs

CLI

AWS CLI

예시 1: 데이터 스토어에 태그 지정

다음 `tag-resource` 코드 예시에서는 데이터 스토어에 태그를 지정합니다.

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

이 명령은 출력을 생성하지 않습니다.

예시 2: 이미지 세트에 태그 지정

다음 `tag-resource` 코드 예시에서는 이미지 세트에 태그를 지정합니다.

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

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS HealthImaging 개발자 안내서의 HealthImaging으로 리소스 태그 지정AWS을 참조하세요 HealthImaging.

- API 세부 정보는 AWS CLI 명령 참조의 TagResource를 참조하세요.

Java

SDK for Java 2.x

```
   public static void tagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Map<String, String> tags) {
       try {
           TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tags(tags)
                   .build();

           medicalImagingClient.tagResource(tagResourceRequest);

           System.out.println("Tags have been added to the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 TagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
```

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tags = {},
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 TagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def tag_resource(self, resource_arn, tags):
        """
        Tag a resource.

        :param resource_arn: The ARN of the resource.
        :param tags: The tags to apply.
        """
        try:
            self.health_imaging_client.tag_resource(resourceArn=resource_arn,
 tags=tags)
        except ClientError as err:
            logger.error(
                "Couldn't tag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 TagResource를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

리소스에 대한 태그 나열

`ListTagsForResource` 작업을 사용하여 AWS HealthImaging의 데이터 스토어 및 이미지 세트에 대한 태그를 나열합니다. 다음 코드 예제에서는 AWS Management Console AWS CLI및 AWS SDKs에서 `ListTagsForResource` 작업을 사용하는 방법을 설명합니다. 자세한 내용은 AWS 일반 참조 가이드의 AWS 리소스 태그 지정을 참조하세요.

리소스의 태그 나열

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열립니다.
3. 세부 정보 탭을 선택하십시오.

   태그 섹션 아래에 모든 데이터 스토어 태그가 나열됩니다.

AWS CLI 및 SDKs

CLI

AWS CLI

예시 1: 데이터 스토어의 리소스 태그 나열

다음 `list-tags-for-resource` 코드 예시에서는 데이터 스토어의 태그를 나열합니다.

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

출력:

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

예시 2: 이미지 세트의 리소스 태그 나열

다음 `list-tags-for-resource` 코드 예시에서는 이미지 세트의 태그를 나열합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 HealthImaging으로 리소스 태그 지정AWS을 참조하세요 HealthImaging.

- API 세부 정보는 AWS CLI Command Reference의 ListTagsForResource를 참조하세요.

Java

SDK for Java 2.x

```
    public static ListTagsForResourceResponse
 listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
            String resourceArn) {
        try {
            ListTagsForResourceRequest listTagsForResourceRequest =
 ListTagsForResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .build();
```

```
            return
 medicalImagingClient.listTagsForResource(listTagsForResourceRequest);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListTagsForResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //     '$metadata': {
```

```
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListTagsForResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def list_tags_for_resource(self, resource_arn):
        """
        List the tags for a resource.

        :param resource_arn: The ARN of the resource.
        :return: The list of tags.
        """
        try:
            tags = self.health_imaging_client.list_tags_for_resource(
                resourceArn=resource_arn
```

```
            )
        except ClientError as err:
            logger.error(
                "Couldn't list tags for resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tags["tags"]
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListTagsForResource를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

리소스의 태그 해제

`UntagResource` 작업을 사용하여 AWS HealthImaging에서 데이터 스토어 및 이미지 세트의 태그를 해제합니다. 다음 코드 예제에서는 AWS Management Console AWS CLI및 AWS SDKs에서 `UntagResource` 작업을 사용하는 방법을 설명합니다. 자세한 내용은 AWS 일반 참조 가이드의 리소스 태그 지정을 참조하세요 AWS.

리소스에서 태그 제거

AWS HealthImaging에 대한 액세스 기본 설정에 따라 메뉴를 선택합니다.

AWS 콘솔

1. HealthImaging 콘솔 데이터 스토어 페이지를 엽니다.
2. 데이터 스토어를 선택합니다.

   데이터 스토어 세부 정보 페이지가 열립니다.
3. 세부 정보 탭을 선택하십시오.
4. 태그 섹션에서 태그 관리를 선택합니다.

   태그 관리 페이지가 열립니다.
5. 제거할 태그 옆에 있는 제거를 선택합니다.
6. Save changes(변경 사항 저장)를 선택합니다.

AWS CLI 및 SDKs

CLI

AWS CLI

예시 1: 데이터 스토어의 태그 해제

다음 `untag-resource` 코드 예시에서는 데이터 스토어의 태그를 해제합니다.

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

이 명령은 출력을 생성하지 않습니다.

예시 2: 이미지 세트의 태그 해제

다음 `untag-resource` 코드 예시에서는 이미지 세트의 태그를 해제합니다.

```
aws medical-imaging untag-resource \
```

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

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS HealthImaging 개발자 안내서의 HealthImaging으로 리소스 태그 지정AWS을 참조하세요 HealthImaging.

- API 세부 정보는 AWS CLI Command Reference의 UntagResource를 참조하세요.

Java

SDK for Java 2.x

```
   public static void untagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Collection<String> tagKeys) {
       try {
           UntagResourceRequest untagResourceRequest =
UntagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tagKeys(tagKeys)
                   .build();

           medicalImagingClient.untagResource(untagResourceRequest);

           System.out.println("Tags have been removed from the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 UntagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 UntagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def untag_resource(self, resource_arn, tag_keys):
        """
        Untag a resource.

        :param resource_arn: The ARN of the resource.
        :param tag_keys: The tag keys to remove.
        """
        try:
            self.health_imaging_client.untag_resource(
                resourceArn=resource_arn, tagKeys=tag_keys
            )
        except ClientError as err:
            logger.error(
                "Couldn't untag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UntagResource를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

> **가용성 예제**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지의 오른쪽 사이드바에 있는 피드백 제공 링크를 사용하여 코드 예제를 요청합니다.

AWS SDKs를 사용한 HealthImaging 코드 예제

다음 코드 예제에서는 AWS 소프트웨어 개발 키트(SDK)와 함께 HealthImaging을 사용하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 관련 시나리오의 컨텍스트에 따라 표시되며, 개별 서비스 함수를 직접적으로 호출하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

시작

HealthImaging 시작

다음은 HealthImaging 사용을 시작하는 방법을 보여주는 코드 예제입니다.

C++

SDK for C++

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS medical-imaging)

# Set this project's name.
project("hello_health-imaging")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})
```

```
if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
    # and set the proper subdirectory to the executable location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_health_imaging.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_health_imaging.cpp 소스 파일의 코드

```
#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;
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
```

```
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

```
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


def hello_medical_imaging(medical_imaging_client):
    """
    Use the AWS SDK for Python (Boto3) to create an AWS HealthImaging
    client and list the data stores in your account.
    This example uses the default settings specified in your shared credentials
    and config files.
```

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


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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListKeyspaces를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

코드 예제

- AWS SDKs를 사용한 HealthImaging의 기본 예제
  - HealthImaging 시작
  - AWS SDKs를 사용한 HealthImaging 작업
    - AWS SDK 또는 CLI와 CopyImageSet 함께 사용
    - AWS SDK 또는 CLI와 CreateDatastore 함께 사용
    - AWS SDK 또는 CLI와 DeleteDatastore 함께 사용

    - AWS SDK 또는 CLI와 DeleteImageSet 함께 사용
    - AWS SDK 또는 CLI와 GetDICOMImportJob 함께 사용
    - AWS SDK 또는 CLI와 GetDatastore 함께 사용
    - AWS SDK 또는 CLI와 GetImageFrame 함께 사용
    - AWS SDK 또는 CLI와 GetImageSet 함께 사용
    - AWS SDK 또는 CLI와 GetImageSetMetadata 함께 사용
    - AWS SDK 또는 CLI와 ListDICOMImportJobs 함께 사용
    - AWS SDK 또는 CLI와 ListDatastores 함께 사용
    - AWS SDK 또는 CLI와 ListImageSetVersions 함께 사용
    - AWS SDK 또는 CLI와 ListTagsForResource 함께 사용
    - AWS SDK 또는 CLI와 SearchImageSets 함께 사용
    - AWS SDK 또는 CLI와 StartDICOMImportJob 함께 사용
    - AWS SDK 또는 CLI와 TagResource 함께 사용
    - AWS SDK 또는 CLI와 UntagResource 함께 사용
    - AWS SDK 또는 CLI와 UpdateImageSetMetadata 함께 사용
- AWS SDKs를 사용한 HealthImaging 시나리오
  - AWS SDK를 사용하여 HealthImaging 이미지 세트 및 이미지 프레임 시작하기
  - AWS SDK를 사용하여 HealthImaging 데이터 스토어에 태그 지정
  - AWS SDK를 사용하여 HealthImaging 이미지 세트에 태그 지정

AWS SDKs를 사용한 HealthImaging의 기본 예제

다음 코드 예제에서는 AWS HealthImaging SDKs에서의 기본 사항을 AWS 사용하는 방법을 보여줍니다.

예시

- HealthImaging 시작
- AWS SDKs를 사용한 HealthImaging 작업
  - AWS SDK 또는 CLI와 CopyImageSet 함께 사용
  - AWS SDK 또는 CLI와 CreateDatastore 함께 사용
  - AWS SDK 또는 CLI와 DeleteDatastore 함께 사용

- AWS SDK 또는 CLI와 DeleteImageSet 함께 사용
- AWS SDK 또는 CLI와 GetDICOMImportJob 함께 사용
- AWS SDK 또는 CLI와 GetDatastore 함께 사용
- AWS SDK 또는 CLI와 GetImageFrame 함께 사용
- AWS SDK 또는 CLI와 GetImageSet 함께 사용
- AWS SDK 또는 CLI와 GetImageSetMetadata 함께 사용
- AWS SDK 또는 CLI와 ListDICOMImportJobs 함께 사용
- AWS SDK 또는 CLI와 ListDatastores 함께 사용
- AWS SDK 또는 CLI와 ListImageSetVersions 함께 사용
- AWS SDK 또는 CLI와 ListTagsForResource 함께 사용
- AWS SDK 또는 CLI와 SearchImageSets 함께 사용
- AWS SDK 또는 CLI와 StartDICOMImportJob 함께 사용
- AWS SDK 또는 CLI와 TagResource 함께 사용
- AWS SDK 또는 CLI와 UntagResource 함께 사용
- AWS SDK 또는 CLI와 UpdateImageSetMetadata 함께 사용

HealthImaging 시작

다음은 HealthImaging 사용을 시작하는 방법을 보여주는 코드 예제입니다.

C++

SDK for C++

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS medical-imaging)

# Set this project's name.
project("hello_health-imaging")

# Set the C++ standard to use to build this target.
```

```
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
    # and set the proper subdirectory to the executable location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_health_imaging.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_health_imaging.cpp 소스 파일의 코드

```
#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;
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

```
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
```

```
def hello_medical_imaging(medical_imaging_client):
    """
    Use the AWS SDK for Python (Boto3) to create an AWS HealthImaging
    client and list the data stores in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

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


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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListKeyspaces를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDKs를 사용한 HealthImaging 작업

다음 코드 예제에서는 AWS SDKs를 사용하여 개별 HealthImaging 작업을 수행하는 방법을 보여줍니다. 각 예제에는 GitHub에 대한 링크가 포함되어 있습니다. 여기에서 코드 설정 및 실행에 대한 지침을 찾을 수 있습니다.

이들 발췌문은 HealthImaging API를 직접적으로 호출하며, 컨텍스트에서 실행되어야 하는 더 큰 프로그램에서 발췌한 코드입니다. AWS SDKs를 사용한 HealthImaging 시나리오 에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 AWS HealthImaging API 참조를 참조하세요.

예시

- AWS SDK 또는 CLI와 CopyImageSet 함께 사용
- AWS SDK 또는 CLI와 CreateDatastore 함께 사용
- AWS SDK 또는 CLI와 DeleteDatastore 함께 사용
- AWS SDK 또는 CLI와 DeleteImageSet 함께 사용
- AWS SDK 또는 CLI와 GetDICOMImportJob 함께 사용
- AWS SDK 또는 CLI와 GetDatastore 함께 사용
- AWS SDK 또는 CLI와 GetImageFrame 함께 사용
- AWS SDK 또는 CLI와 GetImageSet 함께 사용
- AWS SDK 또는 CLI와 GetImageSetMetadata 함께 사용
- AWS SDK 또는 CLI와 ListDICOMImportJobs 함께 사용
- AWS SDK 또는 CLI와 ListDatastores 함께 사용
- AWS SDK 또는 CLI와 ListImageSetVersions 함께 사용
- AWS SDK 또는 CLI와 ListTagsForResource 함께 사용
- AWS SDK 또는 CLI와 SearchImageSets 함께 사용
- AWS SDK 또는 CLI와 StartDICOMImportJob 함께 사용
- AWS SDK 또는 CLI와 TagResource 함께 사용
- AWS SDK 또는 CLI와 UntagResource 함께 사용
- AWS SDK 또는 CLI와 UpdateImageSetMetadata 함께 사용

AWS SDK 또는 CLI와 **CopyImageSet** 함께 사용

다음 코드 예시는 CopyImageSet의 사용 방법을 보여 줍니다.

CLI

AWS CLI

예시 1: 대상 없이 이미지 세트 복사

다음 copy-image-set 예시에서는 대상 없이 이미지 세트의 복제본을 만듭니다.

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

출력:

```
{
    "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"
}
```

예시 2: 대상과 함께 이미지 세트 복사

다음 copy-image-set 예시에서는 대상과 함께 이미지 세트의 복제본을 만듭니다.

```
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"} }'
```

출력:

```
{
    "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"
}
```

예시 3: 소스 이미지 세트의 인스턴스 하위 집합을 대상 이미지 세트로 복사

다음 `copy-image-set` 예시에서는 소스 이미지 세트의 DICOM 인스턴스 하나를 대상 이미지 세트로 복사합니다. 강제 파라미터는 환자, 연구 및 시리즈 수준 속성의 불일치를 재정의하기 위해 제공됩니다.

```
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
```

출력:

```
{
    "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"
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 복사를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CopyImageSet를 참조하세요.

Java

SDK for Java 2.x

```
    /**
     * Copy an AWS HealthImaging image set.
     *
     * @param medicalImagingClient  - The AWS HealthImaging client object.
     * @param datastoreId           - The datastore ID.
     * @param imageSetId            - The image set ID.
```

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

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

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

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

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

           // Optionally designate a destination image set.
           if (destinationImageSetId != null) {
```

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

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

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

복사 가능한 속성을 생성하는 유틸리티 함수입니다.

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 CopyImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

이미지 세트를 복사하는 유틸리티 함수입니다.

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination
 image set.
 * @param {string} destinationVersionId - The optional version ID of the
 destination image set.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "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: 'xxxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
```

```
    //             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);
  }
};
```

대상 없이 이미지 세트를 복사합니다.

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

대상이 있는 이미지 세트를 복사합니다.

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

이미지 세트의 하위 집합을 대상으로 복사하고 강제로 복사합니다.

```
  await copyImageSet(
```

```
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
    "12345678901234567890123456789012",
    "1",
    true,
    ["12345678901234567890123456789012", "11223344556677889900112233445566"],
  );
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 CopyImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

이미지 세트를 복사하는 유틸리티 함수입니다.

```
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"]
```

대상 없이 이미지 세트를 복사합니다.

```
            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,
            )
```

대상이 있는 이미지 세트를 복사합니다.

```
            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,
            )
```

이미지 세트의 하위 집합을 복사합니다.

```
            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,
            )
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CopyImageSet를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **CreateDatastore** 함께 사용

다음 코드 예시는 `CreateDatastore`의 사용 방법을 보여 줍니다.

Bash

AWS CLI Bash 스크립트 사용

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

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

```
local datastore_name response
local option OPTARG # Required to use getopts command in a function.

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

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

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

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

local error_code=${?}

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

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

  echo "$response"

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 CreateDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 생성

다음 `create-datastore` 코드 예시에서는 `my-datastore`라는 데이터 스토어를 생성합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 생성을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CreateDatastore를 참조하세요.

Java

SDK for Java 2.x

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

       return "";
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
```

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 CreateDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CreateDatastore를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DeleteDatastore** 함께 사용

다음 코드 예시는 DeleteDatastore의 사용 방법을 보여 줍니다.

Bash

AWS CLI Bash 스크립트 사용

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

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

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

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

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

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

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

  local error_code=${?}

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

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 삭제

다음 `delete-datastore` 코드 예시에서는 데이터 스토어를 삭제합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteDatastore를 참조하세요.

Java

SDK for Java 2.x

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteDatastore를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DeleteImageSet** 함께 사용

다음 코드 예시는 DeleteImageSet의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 이미지 세트 및 이미지 프레임 시작하기

C++

SDK for C++

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

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

    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 DeleteImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

이미지 세트 삭제

다음 `delete-image-set` 코드 예시에서는 이미지 세트를 삭제합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteImageSet를 참조하세요.

Java

SDK for Java 2.x

```
   public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imagesetId) {
       try {
           DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId)
                   .build();

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
```

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def delete_image_set(self, datastore_id, image_set_id):
        """
        Delete an image set.

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteImageSet를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **GetDICOMImportJob** 함께 사용

다음 코드 예시는 `GetDICOMImportJob`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 이미지 세트 및 이미지 프레임 시작하기

C++

SDK for C++

```
//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
  \param clientConfig: Aws client configuration.
  \return GetDICOMImportJobOutcome: The import job outcome.
*/
Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
AwsDoc::Medical_Imaging::getDICOMImportJob(const Aws::String &dataStoreID,
                                           const Aws::String &importJobID,
                                           const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetDICOMImportJobRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetJobId(importJobID);
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome outcome =
 client.GetDICOMImportJob(
            request);
```

```
    if (!outcome.IsSuccess()) {
        std::cerr << "GetDICOMImportJob error: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome;
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 GetDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

dicom 가져오기 작업의 속성 가져오기

다음 `get-dicom-import-job` 코드 예시에서는 dicom 가져오기 작업의 속성을 가져옵니다.

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

출력:

```
{
    "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/"
    }
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 가져오기 작업 속성 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetDICOMImportJob을 참조하세요.

Java

SDK for Java 2.x

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

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

       return null;
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :param job_id: The ID of the job.
        :return: The job properties.
        """
        try:
            job = self.health_imaging_client.get_dicom_import_job(
                jobId=job_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't get DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobProperties"]
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetDICOMImportJob을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **GetDatastore** 함께 사용

다음 코드 예시는 GetDatastore의 사용 방법을 보여 줍니다.

Bash

AWS CLI Bash 스크립트 사용

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
```

```
  printf "%s\n" "$*" 1>&2
}

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

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

```
  done
  export OPTIND=1

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

  local response

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

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

  echo "$response"

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 GetDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 속성 가져오기

다음 `get-datastore` 코드 예시에서는 데이터 스토어 속성을 가져옵니다.

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

출력:

```
{
    "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"
    }
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 속성 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetDatastore를 참조하세요.

Java

SDK for Java 2.x

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

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

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
```

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetDatastore를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :return: The data store properties.
        """
        try:
            data_store = self.health_imaging_client.get_datastore(
```

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetDatastore를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **GetImageFrame** 함께 사용

다음 코드 예시는 GetImageFrame의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 이미지 세트 및 이미지 프레임 시작하기

C++

SDK for C++

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

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

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

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

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

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

```
    }

    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 GetImageFrame을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

이미지 세트 픽셀 데이터 가져오기

다음 `get-image-frame` 코드 예시에서는 이미지 프레임을 가져옵니다.

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

참고: GetImageFrame 작업이 픽셀 데이터 스트림을 imageframe.jph 파일에 반환하므로 이 코드 예시에는 출력이 포함되지 않습니다. 이미지 프레임 디코딩 및 보기에 대한 자세한 내용은 HTJ2K 디코딩 라이브러리를 참조하세요.

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 픽셀 데이터 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetImageFrame을 참조하세요.

Java

SDK for Java 2.x

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

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

.imageFrameInformation(ImageFrameInformation.builder()

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

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetImageFrame을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

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

```
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetImageFrame을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

        :param file_path_to_write: The path to write the image frame's HTJ2K
 encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.health_imaging_client.get_image_frame(
                datastoreId=datastore_id,
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)
```

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetImageFrame을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **GetImageSet** 함께 사용

다음 코드 예시는 `GetImageSet`의 사용 방법을 보여 줍니다.

CLI

AWS CLI

이미지 세트 속성 가져오기

다음 `get-image-set` 코드 예시에서는 이미지 세트의 속성을 가져옵니다.

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

```
    --image-set-id 18f88ac7870584f58d56256646b4d92b \
    --version-id 1
```

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 속성 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetImageSet를 참조하세요.

Java

SDK for Java 2.x

```
   public static GetImageSetResponse getMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imagesetId,
           String versionId) {
       try {
           GetImageSetRequest.Builder getImageSetRequestBuilder =
GetImageSetRequest.builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId);

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

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

```
            System.exit(1);
        }

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 GetImageSet를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def get_image_set(self, datastore_id, image_set_id, version_id=None):
```

```
        """
        Get the properties of an image set.

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetImageSet를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `GetImageSetMetadata` 함께 사용

다음 코드 예시는 `GetImageSetMetadata`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 이미지 세트 및 이미지 프레임 시작하기

C++

SDK for C++

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

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

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

    return outcome.IsSuccess();
}
```

버전 없이 이미지 세트 메타데이터를 가져옵니다.

```
        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;
        }
```

버전과 함께 이미지 세트 메타데이터를 가져옵니다.

```
        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 세부 정보는 AWS SDK for C++ API 참조의 GetImageSetMetadata를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

예시 1: 버전 없이 이미지 세트 메타데이터 가져오기

다음 `get-image-set-metadata` 코드 예시에서는 버전을 지정하지 않고 이미지 세트의 메타데이터를 가져옵니다.

참고: `outfile`은 필수 파라미터입니다.

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

반환된 메타데이터는 gzip으로 압축되어 studymetadata.json.gz 파일에 저장됩니다. 반환된 JSON 객체의 콘텐츠를 보려면 먼저 압축을 풀어야 합니다.

출력:

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

예시 2: 버전과 함께 이미지 세트 메타데이터 가져오기

다음 `get-image-set-metadata` 코드 예시에서는 지정된 버전의 이미지 세트에 대한 메타데이터를 가져옵니다.

참고: `outfile`은 필수 파라미터입니다.

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

반환된 메타데이터는 gzip으로 압축되어 studymetadata.json.gz 파일에 저장됩니다. 반환된 JSON 객체의 콘텐츠를 보려면 먼저 압축을 풀어야 합니다.

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 메타데이터 가져오기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetImageSetMetadata를 참조하세요.

Java

SDK for Java 2.x

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

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

            if (versionId != null) {
                getImageSetMetadataRequestBuilder =
 getImageSetMetadataRequestBuilder.versionId(versionId);
            }
```

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

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 GetImageSetMetadata를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

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

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

```
  imagesetId = "xxxxxxxxxxxxxx",
  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;
};
```

버전 없이 이미지 세트 메타데이터를 가져옵니다.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
    );
```

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

버전과 함께 이미지 세트 메타데이터를 가져옵니다.

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

- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetImageSetMetadata를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

이미지 세트 메타데이터를 가져오는 유틸리티 함수입니다.

```
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
```

버전 없이 이미지 세트 메타데이터를 가져옵니다.

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

버전과 함께 이미지 세트 메타데이터를 가져옵니다.

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetImageSetMetadata를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **`ListDICOMImportJobs`** 함께 사용

다음 코드 예시는 `ListDICOMImportJobs`의 사용 방법을 보여 줍니다.

CLI

AWS CLI

dicom 가져오기 작업 나열

다음 `list-dicom-import-jobs` 코드 예시에서는 dicom 가져오기 작업을 나열합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 가져오기 작업 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListDICOMImportJobs를 참조하세요.

Java

SDK for Java 2.x

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

        try {
            ListDicomImportJobsRequest listDicomImportJobsRequest =
 ListDicomImportJobsRequest.builder()
```

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

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

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListDICOMImportJobs를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

  const commandParams = { datastoreId: datastoreId };
```

```
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

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

  return jobSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListDICOMImportJobs를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def list_dicom_import_jobs(self, datastore_id):
        """
        List the DICOM import jobs.

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListDICOMImportJobs를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `ListDatastores` 함께 사용

다음 코드 예시는 `ListDatastores`의 사용 방법을 보여 줍니다.

Bash

AWS CLI Bash 스크립트 사용

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

###############################################################################
# function imaging_list_datastores
#
# List the HealthImaging data stores in the account.
#
# Returns:
#       [[datastore_name, datastore_id, datastore_status]]
#    And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function imaging_list_datastores() {
  local option OPTARG # Required to use getopts command in a function.
```

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

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

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

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

  echo "$response"

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

데이터 스토어 나열

다음 `list-datastores` 코드 예시에서는 사용 가능한 데이터 스토어를 나열합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 데이터 스토어 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListDatastores를 참조하세요.

Java

SDK for Java 2.x

```
    public static List<DatastoreSummary>
 listMedicalImagingDatastores(MedicalImagingClient medicalImagingClient) {
```

```
        try {
            ListDatastoresRequest datastoreRequest =
 ListDatastoresRequest.builder()
                    .build();
            ListDatastoresIterable responses =
 medicalImagingClient.listDatastoresPaginator(datastoreRequest);
            List<DatastoreSummary> datastoreSummaries = new ArrayList<>();

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

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

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

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

  return datastoreSummaries;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListDatastores를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListKeyspaces를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `ListImageSetVersions` 함께 사용

다음 코드 예시는 `ListImageSetVersions`의 사용 방법을 보여 줍니다.

CLI

AWS CLI

이미지 세트 버전 나열

다음 `list-image-set-versions` 코드 예시에서는 이미지 세트의 버전 기록을 나열합니다.

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

출력:

```
{
    "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
        }
    ]
}
```

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 버전 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListImageSetVersions를 참조하세요.

Java

SDK for Java 2.x

```
    public static List<ImageSetProperties>
 listMedicalImageSetVersions(MedicalImagingClient medicalImagingClient,
            String datastoreId,
            String imagesetId) {
        try {
```

```
            ListImageSetVersionsRequest getImageSetRequest =
ListImageSetVersionsRequest.builder()
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId)
                    .build();

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

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

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListImageSetVersions를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

```
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

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

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListImageSetVersions를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

```
            raise
        else:
            return image_set_properties_list
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListImageSetVersions를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `ListTagsForResource` 함께 사용

다음 코드 예시는 `ListTagsForResource`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 데이터 저장소에 태그 지정
- 이미지 세트 태그 지정

CLI

AWS CLI

예제 1: 데이터 스토어에 대한 리소스 태그 나열

다음 `list-tags-for-resource` 코드 예시에서는 데이터 스토어의 태그를 나열합니다.

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

출력:

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

예시 2: 이미지 세트의 리소스 태그 나열

다음 `list-tags-for-resource` 코드 예시에서는 이미지 세트의 태그를 나열합니다.

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 HealthImaging을 사용하여 리소스 태그 지정AWS 을 참조하세요 HealthImaging.

- API 세부 정보는 AWS CLI Command Reference의 ListTagsForResource를 참조하세요.

Java

SDK for Java 2.x

```
    public static ListTagsForResourceResponse
 listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
            String resourceArn) {
```

```
        try {
            ListTagsForResourceRequest listTagsForResourceRequest =
 ListTagsForResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .build();

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

        return null;
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 ListTagsForResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi",
) => {
  const response = await medicalImagingClient.send(
```

```
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 ListTagsForResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


    def list_tags_for_resource(self, resource_arn):
        """
        List the tags for a resource.

        :param resource_arn: The ARN of the resource.
```

```
        :return: The list of tags.
        """
        try:
            tags = self.health_imaging_client.list_tags_for_resource(
                resourceArn=resource_arn
            )
        except ClientError as err:
            logger.error(
                "Couldn't list tags for resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tags["tags"]
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListTagsForResource를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `SearchImageSets` 함께 사용

다음 코드 예시는 `SearchImageSets`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 이미지 세트 및 이미지 프레임 시작하기

C++

SDK for C++

이미지 세트 검색을 위한 유틸리티 함수.

```
//! 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;
}
```

사용 사례 #1: EQUAL 연산자.

```
        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;
            }
        }
```

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

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

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

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

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

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

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

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

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

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

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
        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;
            }
        }
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
        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;
           }
       }
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 SearchImageSets를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

예시 1: EQUAL 연산자로 이미지 세트 검색

다음 `search-image-sets` 코드 예시에서는 EQUAL 연산자를 사용하여 특정 값을 기준으로 이미지 세트를 검색합니다.

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

`search-criteria.json`의 콘텐츠

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

출력:

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

예시 2: DICOMStudyDate 및 DICOMStudyTime을 사용하여 BETWEEN 연산자로 이미지 세트 검색

다음 `search-image-sets` 코드 예시에서는 1990년 1월 1일(오전 12시)에서 2023년 1월 1일(오전 12시) 사이에 생성된 DICOM 연구가 있는 이미지 세트를 검색합니다.

참고: DICOMStudyTime은 선택 사항입니다. 해당 날짜가 없는 경우 필터링에 제공되는 날짜의 시간 값은 오전 12시(하루의 시작)입니다.

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

`search-criteria.json`의 콘텐츠

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

```
        }],
        "operator": "BETWEEN"
    }]
}
```

출력:

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

예시 3: CreatedAt을 사용하여 BETWEEN 연산자로 이미지 세트 검색(시간 연구가 이전에 지속됨)

다음 `search-image-sets` 코드 예시에서는 UTC 시간대의 시간 범위에서 HealthImaging에 대한 DICOM 연구가 지속된 이미지 세트를 검색합니다.

참고: createdAt을 예시 형식('1985-04-12T23:20:50.52Z')으로 제공합니다.

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

`search-criteria.json`의 콘텐츠

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

출력:

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

예시 4: updatedAt의 DICOMSeriesInstanceUID 및 BETWEEN에서 EQUAL 연산자로 이미지 세트를 검색하고 updatedAt 필드의 ASC 순서로 응답 정렬

다음 `search-image-sets` 코드 예시에서는 DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN을 사용하여 이미지 세트를 검색하고 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

참고: updatedAt을 예시 형식('1985-04-12T23:20:50.52Z')으로 제공합니다.

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

search-criteria.json의 콘텐츠

```
{
    "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"
    }
}
```

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 검색을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 SearchImageSets를 참조하세요.

Java

SDK for Java 2.x

이미지 세트 검색을 위한 유틸리티 함수.

```
    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;
```

```
    }
```

사용 사례 #1: EQUAL 연산자.

```
        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();
        }
```

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

```
        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();
        }
```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
        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();
        }
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
        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();
        }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 SearchImageSets를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

이미지 세트 검색을 위한 유틸리티 함수.

```
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;
};
```

사용 사례 #1: EQUAL 연산자.

```
  const datastoreId = "12345678901234567890123456789012";

  try {
    const searchCriteria = {
```

```
      filters: [
        {
          values: [{ DICOMPatientId: "1234567" }],
          operator: "EQUAL",
        },
      ],
    };

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

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

```
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);
}
```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
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);
}
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
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);
  }
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 SearchImageSets를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

이미지 세트 검색을 위한 유틸리티 함수.

```
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
```

사용 사례 #1: EQUAL 연산자.

```
        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}")
```

사용 사례 #2: DICOMStudyDate 및 DICOMStudyTime을 사용한 BETWEEN 연산자.

```
        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}"
        )
```

사용 사례 #3: createdAt을 사용한 BETWEEN 연산자. 시간 연구가 이전에 지속되었습니다.

```
        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}"
        )
```

사용 사례 #4: DICOMSeriesInstanceUID의 EQUAL 연산자와 updatedAt의 BETWEEN 및 updatedAt 필드의 ASC 순서로 응답을 정렬합니다.

```
        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}")
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 SearchImageSets를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **StartDICOMImportJob** 함께 사용

다음 코드 예시는 `StartDICOMImportJob`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 이미지 세트 및 이미지 프레임 시작하기

C++

SDK for C++

```
//! Routine which starts a HealthImaging import job.
```

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

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

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

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

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 StartDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CLI

AWS CLI

dicom 가져오기 작업 시작

다음 `start-dicom-import-job` 코드 예시에서는 dicom 가져오기 작업을 시작합니다.

```
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"
```

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 가져오기 작업 시작을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 StartDICOMImportJob을 참조하세요.

Java

SDK for Java 2.x

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

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

       return "";
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 StartDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 StartDICOMImportJob을 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

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

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

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 StartTranscriptionJob을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `TagResource` 함께 사용

다음 코드 예시는 `TagResource`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 데이터 저장소에 태그 지정
- 이미지 세트 태그 지정

CLI

AWS CLI

예제 1: 데이터 스토어에 태그 지정

다음 `tag-resource` 코드 예시에서는 데이터 스토어에 태그를 지정합니다.

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

이 명령은 출력을 생성하지 않습니다.

예시 2: 이미지 세트에 태그 지정

다음 `tag-resource` 코드 예시에서는 이미지 세트에 태그를 지정합니다.

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

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS HealthImaging 개발자 안내서의 HealthImaging을 사용하여 리소스 태그 지정AWS 을 참조하세요 HealthImaging.

- API 세부 정보는 AWS CLI 명령 참조의 TagResource를 참조하세요.

Java

SDK for Java 2.x

```
    public static void tagMedicalImagingResource(MedicalImagingClient
 medicalImagingClient,
            String resourceArn,
            Map<String, String> tags) {
        try {
            TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .tags(tags)
                    .build();

            medicalImagingClient.tagResource(tagResourceRequest);

            System.out.println("Tags have been added to the resource.");
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 TagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
```

```
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tags = {},
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 TagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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

```
    def tag_resource(self, resource_arn, tags):
        """
        Tag a resource.

        :param resource_arn: The ARN of the resource.
        :param tags: The tags to apply.
        """
        try:
            self.health_imaging_client.tag_resource(resourceArn=resource_arn,
tags=tags)
        except ClientError as err:
            logger.error(
                "Couldn't tag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 TagResource를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `UntagResource` 함께 사용

다음 코드 예시는 `UntagResource`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 데이터 저장소에 태그 지정
- 이미지 세트 태그 지정

CLI

AWS CLI

예제 1: 데이터 스토어의 태그 해제

다음 `untag-resource` 코드 예시에서는 데이터 스토어의 태그를 해제합니다.

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

이 명령은 출력을 생성하지 않습니다.

예시 2: 이미지 세트의 태그 해제

다음 `untag-resource` 코드 예시에서는 이미지 세트의 태그를 해제합니다.

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

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS HealthImaging 개발자 안내서의 HealthImaging을 사용하여 리소스 태그 지정AWS 을 참조하세요 HealthImaging.

- API 세부 정보는 AWS CLI Command Reference의 UntagResource를 참조하세요.

Java

SDK for Java 2.x

```
   public static void untagMedicalImagingResource(MedicalImagingClient
medicalImagingClient,
           String resourceArn,
           Collection<String> tagKeys) {
       try {
           UntagResourceRequest untagResourceRequest =
UntagResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .tagKeys(tagKeys)
                   .build();

           medicalImagingClient.untagResource(untagResourceRequest);

           System.out.println("Tags have been removed from the resource.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 UntagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

/**
```

```
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 UntagResource를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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

```
        self.health_imaging_client = health_imaging_client


    def untag_resource(self, resource_arn, tag_keys):
        """
        Untag a resource.

        :param resource_arn: The ARN of the resource.
        :param tag_keys: The tag keys to remove.
        """
        try:
            self.health_imaging_client.untag_resource(
                resourceArn=resource_arn, tagKeys=tag_keys
            )
        except ClientError as err:
            logger.error(
                "Couldn't untag resource. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UntagResource를 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **UpdateImageSetMetadata** 함께 사용

다음 코드 예시는 UpdateImageSetMetadata의 사용 방법을 보여 줍니다.

CLI

AWS CLI

예시 1: 이미지 세트 메타데이터에 속성을 삽입하거나 업데이트

다음 update-image-set-metadata 예시에서는 이미지 세트 메타데이터의 속성을 삽입하거나 업데이트합니다.

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json의 콘텐츠

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

출력:

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

예시 2: 이미지 세트 메타데이터에서 속성 제거

다음 update-image-set-metadata 예시에서는 이미지 세트 메타데이터에서 속성을 제거합니다.

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json의 콘텐츠

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

출력:

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

예시 3: 이미지 세트 메타데이터에서 인스턴스 제거

다음 update-image-set-metadata 예시에서는 이미지 세트 메타데이터에서 인스턴스를 제거합니다.

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --cli-binary-format raw-in-base64-out \
```

```
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json의 콘텐츠

```
{
    "DICOMUpdates": {
        "removableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\":
 {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}"
    }
}
```

출력:

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

예시 4: 이미지 세트를 이전 버전으로 되돌리기

다음 update-image-set-metadata 예시에서는 이미지 세트를 이전 버전으로 되돌리는 방법을 보여줍니다. CopyImageSet 및 UpdateImageSetMetadata 작업은 새 버전의 이미지 세트를 생성합니다.

```
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"}'
```

출력:

```
{
```

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

예시 5: 인스턴스에 프라이빗 DICOM 데이터 요소 추가

다음 `update-image-set-metadata` 예시에서는 이미지 세트 내 지정된 인스턴스에 프라이빗 요소를 추가하는 방법을 보여줍니다. DICOM 표준은 표준 데이터 요소에 포함할 수 없는 정보를 통신하기 위한 프라이빗 데이터 요소를 허용합니다. UpdateImageSetMetadata 작업을 사용하여 프라이빗 데이터 요소를 생성, 업데이트 및 삭제할 수 있습니다.

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

`metadata-updates.json`의 콘텐츠

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":
 {\"001910F9\": \"97\"},\"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}"
    }
}
```

출력:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
```

```
    "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

예시 6: 프라이빗 DICOM 데이터 요소를 인스턴스로 업데이트

다음 `update-image-set-metadata` 예시에서는 이미지 세트 내 인스턴스에 속하는 프라이빗 데이터 요소의 값을 업데이트하는 방법을 보여줍니다.

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

`metadata-updates.json`의 콘텐츠

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

출력:

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

예시 7: SOPInstanceUID를 강제 파라미터로 업데이트

다음 `update-image-set-metadata` 예시에서는 강제 파라미터를 사용하여 DICOM 메타데이터 제약 조건을 재정의하여 SOPInstanceUID를 업데이트하는 방법을 보여줍니다.

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

`metadata-updates.json`의 콘텐츠

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

출력:

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

자세한 내용은 AWS HealthImaging 개발자 안내서의 이미지 세트 메타데이터 업데이트를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateImageSetMetadata를 참조하세요.

Java

SDK for Java 2.x

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

           UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

```
        }
    }
```

사용 사례 #1: 속성을 삽입하거나 업데이트합니다.

```
                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);
```

사용 사례 #2: 속성을 제거합니다.

```
                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);
```

사용 사례 #3: 인스턴스를 제거합니다.

```
                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);
```

사용 사례 #4: 이전 버전으로 되돌립니다.

```
                // 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);
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateImageSetMetadata를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

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

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

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

사용 사례 #1: 속성을 삽입 또는 업데이트하고 강제로 업데이트합니다.

```
    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,
  );
```

사용 사례 #2: 속성을 제거합니다.

```
  // 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,
  );
```

사용 사례 #3: 인스턴스를 제거합니다.

```
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,
);
```

사용 사례 #4: 이전 버전으로 되돌립니다.

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

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
```

```
      versionID,
      updateMetadata,
    );
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateImageSetMetadata를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param metadata: The image set metadata as a dictionary.
            For example {"DICOMUpdates": {"updatableAttributes":
            "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{\"PatientName\":
\"Garcia^Gloria\"}}}"}}
        :param: force: Force the update.
        :return: The updated image set metadata.
        """
        try:
            updated_metadata =
 self.health_imaging_client.update_image_set_metadata(
```

```
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                latestVersionId=version_id,
                updateImageSetMetadataUpdates=metadata,
                force=force,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update image set metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return updated_metadata
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

사용 사례 #1: 속성을 삽입하거나 업데이트합니다.

```
            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
            )
```

사용 사례 #2: 속성을 제거합니다.

```
            # 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
            )
```

사용 사례 #3: 인스턴스를 제거합니다.

```
            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
            )
```

사용 사례 #4: 이전 버전으로 되돌립니다.

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

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateImageSetMetadata을 참조하십시오.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDKs를 사용한 HealthImaging 시나리오

다음 코드 예제에서는 HealthImaging AWS SDKs에서 일반적인 시나리오를 구현하는 방법을 보여줍니다. 이러한 시나리오에서는 HealthImaging 내에서 여러 함수를 호출하거나 다른 함수와 결합하여 특정 작업을 수행하는 방법을 보여줍니다 AWS 서비스. 각 시나리오에는 전체 소스 코드에 대한 링크가 포함되어 있습니다. 여기에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

시나리오는 컨텍스트에 맞는 서비스 작업을 이해하는 데 도움이 되도록 중급 수준의 경험을 대상으로 합니다.

예시

- AWS SDK를 사용하여 HealthImaging 이미지 세트 및 이미지 프레임 시작하기
- AWS SDK를 사용하여 HealthImaging 데이터 스토어에 태그 지정
- AWS SDK를 사용하여 HealthImaging 이미지 세트에 태그 지정

AWS SDK를 사용하여 HealthImaging 이미지 세트 및 이미지 프레임 시작하기

다음 코드 예제에서는 HealthImaging에서 DICOM 파일을 가져오고 이미지 프레임을 다운로드하는 방법을 보여줍니다.

구현은 명령줄 애플리케이션으로 구성됩니다.

- DICOM 가져오기의 리소스를 설정합니다.
- 데이터 스토어로 DICOM 파일을 가져옵니다.
- 가져오기 작업의 이미지 세트 ID를 검색합니다.
- 이미지 세트의 이미지 프레임 ID를 검색합니다.
- 이미지 프레임을 다운로드, 디코딩 및 확인합니다.
- 리소스를 정리합니다.

C++

SDK for C++

필요한 리소스를 사용하여 AWS CloudFormation 스택을 생성합니다.

```
    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: ");
    }
```

Amazon S3 가져오기 버킷에 DICOM 파일을 복사합니다.

```
    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;
   }
```

Amazon S3 데이터 스토어로 DICOM 파일을 가져옵니다.

```
bool AwsDoc::Medical_Imaging::startDicomImport(const Aws::String &dataStoreID,
```

```
                                                const Aws::String
 &inputBucketName,
                                                const Aws::String &inputDirectory,
                                                const Aws::String
 &outputBucketName,
                                                const Aws::String
 &outputDirectory,
                                                const Aws::String &roleArn,
                                                Aws::String &importJobId,
                                                const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    bool result = false;
    if (startDICOMImportJob(dataStoreID, inputBucketName, inputDirectory,
                            outputBucketName, outputDirectory, roleArn,
 importJobId,
                            clientConfiguration)) {
        std::cout << "DICOM import job started with job ID " << importJobId <<
 "."
                  << std::endl;
        result = waitImportJobCompleted(dataStoreID, importJobId,
 clientConfiguration);
        if (result) {
            std::cout << "DICOM import job completed." << std::endl;

        }
    }

    return result;
}

//! Routine which starts a HealthImaging import job.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param inputBucketName: The name of the Amazon S3 bucket containing the DICOM
 files.
  \param inputDirectory: The directory in the S3 bucket containing the DICOM
 files.
  \param outputBucketName: The name of the S3 bucket for the output.
  \param outputDirectory: The directory in the S3 bucket to store the output.
  \param roleArn: The ARN of the IAM role with permissions for the import.
  \param importJobId: A string to receive the import job ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
```

```
bool AwsDoc::Medical_Imaging::startDICOMImportJob(
        const Aws::String &dataStoreID, const Aws::String &inputBucketName,
        const Aws::String &inputDirectory, const Aws::String &outputBucketName,
        const Aws::String &outputDirectory, const Aws::String &roleArn,
        Aws::String &importJobId,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(clientConfig);
    Aws::String inputURI = "s3://" + inputBucketName + "/" + inputDirectory +
 "/";
    Aws::String outputURI = "s3://" + outputBucketName + "/" + outputDirectory +
 "/";
    Aws::MedicalImaging::Model::StartDICOMImportJobRequest
 startDICOMImportJobRequest;
    startDICOMImportJobRequest.SetDatastoreId(dataStoreID);
    startDICOMImportJobRequest.SetDataAccessRoleArn(roleArn);
    startDICOMImportJobRequest.SetInputS3Uri(inputURI);
    startDICOMImportJobRequest.SetOutputS3Uri(outputURI);

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

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

    return startDICOMImportJobOutcome.IsSuccess();
}


//! Routine which waits for a DICOM import job to complete.
/*!
 * @param dataStoreID: The HealthImaging data store ID.
 * @param importJobId: The import job ID.
 * @param clientConfiguration : Aws client configuration.
 * @return  bool: Function succeeded.
 */
bool AwsDoc::Medical_Imaging::waitImportJobCompleted(const Aws::String
 &datastoreID,
```

```
                                                        const Aws::String
 &importJobId,
                                                        const
 Aws::Client::ClientConfiguration &clientConfiguration) {

    Aws::MedicalImaging::Model::JobStatus jobStatus =
 Aws::MedicalImaging::Model::JobStatus::IN_PROGRESS;
    while (jobStatus == Aws::MedicalImaging::Model::JobStatus::IN_PROGRESS) {
        std::this_thread::sleep_for(std::chrono::seconds(1));

        Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
 getDicomImportJobOutcome = getDICOMImportJob(
                datastoreID, importJobId,
                clientConfiguration);

        if (getDicomImportJobOutcome.IsSuccess()) {
            jobStatus =
 getDicomImportJobOutcome.GetResult().GetJobProperties().GetJobStatus();

            std::cout << "DICOM import job status: " <<

 Aws::MedicalImaging::Model::JobStatusMapper::GetNameForJobStatus(
                              jobStatus) << std::endl;
        }
        else {
            std::cerr << "Failed to get import job status because "
                      << getDicomImportJobOutcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    }

    return jobStatus == Aws::MedicalImaging::Model::JobStatus::COMPLETED;
}

//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
  \param clientConfig: Aws client configuration.
  \return GetDICOMImportJobOutcome: The import job outcome.
*/
Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
AwsDoc::Medical_Imaging::getDICOMImportJob(const Aws::String &dataStoreID,
```

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

    return outcome;
}
```

DICOM 가져오기 작업으로 생성된 이미지 세트를 가져옵니다.

```
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;
```

```
}
```

이미지 세트의 이미지 프레임 정보를 가져옵니다.

```
bool AwsDoc::Medical_Imaging::getImageFramesForImageSet(const Aws::String
 &dataStoreID,
                                                         const Aws::String
 &imageSetID,
                                                         const Aws::String
 &outDirectory,

 Aws::Vector<ImageFrameInfo> &imageFrames,
                                                         const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::String fileName = outDirectory + "/" + imageSetID +
 "_metadata.json.gzip";
    bool result = false;
    if (getImageSetMetadata(dataStoreID, imageSetID, "", // Empty string for
 version ID.
                            fileName, clientConfiguration)) {
        try {
            std::string metadataGZip;
            {
                std::ifstream inFileStream(fileName.c_str(), std::ios::binary);
                if (!inFileStream) {
                    throw std::runtime_error("Failed to open file " + fileName);
                }

                std::stringstream stringStream;
                stringStream << inFileStream.rdbuf();
                metadataGZip = stringStream.str();
            }
            std::string metadataJson = gzip::decompress(metadataGZip.data(),
                                                         metadataGZip.size());
            // Use JMESPath to extract the image set IDs.
            // https://jmespath.org/specification.html
            jsoncons::json doc = jsoncons::json::parse(metadataJson);
            std::string jmesPathExpression = "Study.Series.*.Instances[].*[]";
            jsoncons::json instances = jsoncons::jmespath::search(doc,

 jmesPathExpression);
            for (auto &instance: instances.array_range()) {
```

```
                jmesPathExpression = "DICOM.RescaleSlope";
                std::string rescaleSlope = jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();
                jmesPathExpression = "DICOM.RescaleIntercept";
                std::string rescaleIntercept =
jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();

                jmesPathExpression = "ImageFrames[][]";
                jsoncons::json imageFramesJson =
jsoncons::jmespath::search(instance,

jmesPathExpression);

                for (auto &imageFrame: imageFramesJson.array_range()) {
                    ImageFrameInfo imageFrameIDs;
                    imageFrameIDs.mImageSetId = imageSetID;
                    imageFrameIDs.mImageFrameId = imageFrame.find(
                            "ID")->value().as_string();
                    imageFrameIDs.mRescaleIntercept = rescaleIntercept;
                    imageFrameIDs.mRescaleSlope = rescaleSlope;
                    imageFrameIDs.MinPixelValue = imageFrame.find(
                            "MinPixelValue")->value().as_string();
                    imageFrameIDs.MaxPixelValue = imageFrame.find(
                            "MaxPixelValue")->value().as_string();

                    jmesPathExpression =
"max_by(PixelDataChecksumFromBaseToFullResolution, &Width).Checksum";
                    jsoncons::json checksumJson =
jsoncons::jmespath::search(imageFrame,

jmesPathExpression);
                    imageFrameIDs.mFullResolutionChecksum =
checksumJson.as_integer<uint32_t>();

                    imageFrames.emplace_back(imageFrameIDs);
                }
            }

            result = true;
        }
        catch (const std::exception &e) {
```

```
            std::cerr << "getImageFramesForImageSet failed because " << e.what()
                      << std::endl;
        }
    }

    return result;
}

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

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

이미지 프레임을 다운로드, 디코딩 및 확인합니다.

```
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;
}
```

리소스를 정리합니다.

```
bool AwsDoc::Medical_Imaging::cleanup(const Aws::String &stackName,
                                      const Aws::String &dataStoreId,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    bool result = true;

    if (!stackName.empty() && askYesNoQuestion(
            "Would you like to delete the stack " + stackName + "? (y/n)")) {
        std::cout << "Deleting the image sets in the stack." << std::endl;
        result &= emptyDatastore(dataStoreId, clientConfiguration);
        printAsterisksLine();
        std::cout << "Deleting the stack." << std::endl;
        result &= deleteStack(stackName, clientConfiguration);
    }
    return result;
}

bool AwsDoc::Medical_Imaging::emptyDatastore(const Aws::String &datastoreID,
                                             const
 Aws::Client::ClientConfiguration &clientConfiguration) {

    Aws::MedicalImaging::Model::SearchCriteria emptyCriteria;
    Aws::Vector<Aws::String> imageSetIDs;
    bool result = false;
    if (searchImageSets(datastoreID, emptyCriteria, imageSetIDs,
                        clientConfiguration)) {
        result = true;
        for (auto &imageSetID: imageSetIDs) {
            result &= deleteImageSet(datastoreID, imageSetID,
 clientConfiguration);
        }
    }

    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 다음 주제를 참조하세요.
  - DeleteImageSet
  - GetDICOMImportJob

- GetImageFrame
- GetImageSetMetadata
- SearchImageSets
- StartDICOMImportJob

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

단계(index.js)를 오케스트레이션합니다.

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
```

```
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
```

```
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Health Imaging Workflow",
    description:
      "Work with DICOM images using an AWS Health Imaging data store.",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes]
 [-v|--verbose]",
  });
}
```

리소스(deploy-steps.js)를 배포합니다.

```
import fs from "node:fs/promises";
```

```
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

```
export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
```

```
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
```

```
);
```

DICOM 파일(dataset-steps.js)을 복사합니다.

```
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.`,
);
```

데이터 스토어(import-steps.js)로 가져오기를 시작합니다.

```
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}`,
);
```

이미지 세트 IDs 가져옵니다.image-set-steps.js

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
[] } }
 * }} State
```

```
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce((ids, next) => {
        return Object.assign({}, ids, {
          [next.imageSetId]: next.imageSetId,
        });
      }, {});
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

이미지 프레임 IDs 가져옵니다.image-frame-steps.js

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node:util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
```

```
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
```

```
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
);
```

이미지 프레임(verify-steps.js)을 확인합니다. AWS HealthImaging Pixel 데이터 확인 라이브러리가 확인에 사용되었습니다.

```
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
```

```
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
    default: true,
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (/** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;
```

```
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid}
 and sop ${sopInstanceUid}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(
                    `Verification tool exited with code ${code} for image set
 ${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

리소스를 폐기합니다(clean-up-steps.js).

```
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */
```

```
/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;
```

```
    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - DeleteImageSet

- GetDICOMImportJob
- GetImageFrame
- GetImageSetMetadata
- SearchImageSets
- StartDICOMImportJob

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

필요한 리소스를 사용하여 AWS CloudFormation 스택을 생성합니다.

```
    def deploy(self):
        """
        Deploys prerequisite resources used by the scenario. The resources are
        defined in the associated `setup.yaml` AWS CloudFormation script and are
 deployed
        as a CloudFormation stack, so they can be easily managed and destroyed.
        """

        print("\t\tLet's deploy the stack for resource creation.")
        stack_name = q.ask("\t\tEnter a name for the stack: ", q.non_empty)

        data_store_name = q.ask(
            "\t\tEnter a name for the Health Imaging Data Store: ", q.non_empty
        )

        account_id = boto3.client("sts").get_caller_identity()["Account"]

        with open(
            "../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml"
        ) as setup_file:
            setup_template = setup_file.read()
        print(f"\t\tCreating {stack_name}.")
```

```
        stack = self.cf_resource.create_stack(
            StackName=stack_name,
            TemplateBody=setup_template,
            Capabilities=["CAPABILITY_NAMED_IAM"],
            Parameters=[
                {
                    "ParameterKey": "datastoreName",
                    "ParameterValue": data_store_name,
                },
                {
                    "ParameterKey": "userAccountID",
                    "ParameterValue": account_id,
                },
            ],
        )
        print("\t\tWaiting for stack to deploy. This typically takes a minute or
two.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
        waiter.wait(StackName=stack.name)
        stack.load()
        print(f"\t\tStack status: {stack.stack_status}")

        outputs_dictionary = {
            output["OutputKey"]: output["OutputValue"] for output in
stack.outputs
        }
        self.input_bucket_name = outputs_dictionary["BucketName"]
        self.output_bucket_name = outputs_dictionary["BucketName"]
        self.role_arn = outputs_dictionary["RoleArn"]
        self.data_store_id = outputs_dictionary["DatastoreID"]
        return stack
```

Amazon S3 가져오기 버킷에 DICOM 파일을 복사합니다.

```
    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.")
```

Amazon S3 데이터 스토어로 DICOM 파일을 가져옵니다.

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def start_dicom_import_job(
        self,
        data_store_id,
        input_bucket_name,
        input_directory,
        output_bucket_name,
        output_directory,
        role_arn,
    ):
        """
        Routine which starts a HealthImaging import job.

        :param data_store_id: The HealthImaging data store ID.
        :param input_bucket_name: The name of the Amazon S3 bucket containing the
 DICOM files.
        :param input_directory: The directory in the S3 bucket containing the
 DICOM files.
        :param output_bucket_name: The name of the S3 bucket for the output.
        :param output_directory: The directory in the S3 bucket to store the
 output.
        :param role_arn: The ARN of the IAM role with permissions for the import.
        :return: The job ID of the import.
        """

        input_uri = f"s3://{input_bucket_name}/{input_directory}/"
        output_uri = f"s3://{output_bucket_name}/{output_directory}/"
```

```
        try:
            job = self.medical_imaging_client.start_dicom_import_job(
                jobName="examplejob",
                datastoreId=data_store_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_uri,
                outputS3Uri=output_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

DICOM 가져오기 작업으로 생성된 이미지 세트를 가져옵니다.

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
```

```
        """
        Retrieves the image sets created for an import job.

        :param datastore_id: The HealthImaging data store ID
        :param import_job_id: The import job ID
        :return: List of image set IDs
        """

        import_job = self.medical_imaging_client.get_dicom_import_job(
            datastoreId=datastore_id, jobId=import_job_id
        )

        output_uri = import_job["jobProperties"]["outputS3Uri"]

        bucket = output_uri.split("/")[2]
        key = "/".join(output_uri.split("/")[3:])

        # Try to get the manifest.
        retries = 3
        while retries > 0:
            try:
                obj = self.s3_client.get_object(
                    Bucket=bucket, Key=key + "job-output-manifest.json"
                )
                body = obj["Body"]
                break
            except ClientError as error:
                retries = retries - 1
                time.sleep(3)
        try:
            data = json.load(body)
            expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
            image_sets = expression.search(data)
        except json.decoder.JSONDecodeError as error:
            image_sets = import_job["jobProperties"]

        return image_sets


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

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

이미지 세트의 이미지 프레임 정보를 가져옵니다.

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client
```

```
    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_frames_for_image_set(self, datastore_id, image_set_id,
out_directory):
        """
        Get the image frames for an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param out_directory: The directory to save the file.
        :return: The image frames.
        """
        image_frames = []
        file_name = os.path.join(out_directory,
f"{image_set_id}_metadata.json.gzip")
        file_name = file_name.replace("/", "\\\\")
        self.get_image_set_metadata(file_name, datastore_id, image_set_id)
        try:
            with gzip.open(file_name, "rb") as f_in:
                doc = json.load(f_in)
            instances = jmespath.search("Study.Series.*.Instances[].*[]", doc)
            for instance in instances:
                rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
                rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)
                image_frames_json = jmespath.search("ImageFrames[][]", instance)
                for image_frame in image_frames_json:
                    checksum_json = jmespath.search(
                        "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",
                        image_frame,
                    )
                    image_frame_info = {
                        "imageSetId": image_set_id,
                        "imageFrameId": image_frame["ID"],
                        "rescaleIntercept": rescale_intercept,
                        "rescaleSlope": rescale_slope,
                        "minPixelValue": image_frame["MinPixelValue"],
                        "maxPixelValue": image_frame["MaxPixelValue"],
```

```
                        "fullResolutionChecksum": checksum_json["Checksum"],
                    }
                    image_frames.append(image_frame_info)
            return image_frames
        except TypeError:
            return {}
        except ClientError as err:
            logger.error(
                "Couldn't get image frames for image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        return image_frames


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

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """

        try:
            if version_id:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
            with open(metadata_file, "wb") as f:
                for chunk in
image_set_metadata["imageSetMetadataBlob"].iter_chunks():
```

```
                    if chunk:
                        f.write(chunk)

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

이미지 프레임을 다운로드, 디코딩 및 확인합니다.

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


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

        :param file_path_to_write: The path to write the image frame's HTJ2K
 encoded pixel data.
```

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


    def download_decode_and_check_image_frames(
        self, data_store_id, image_frames, out_directory
    ):
        """
        Downloads image frames, decodes them, and uses the checksum to validate
        the decoded images.

        :param data_store_id: The HealthImaging data store ID.
        :param image_frames: A list of dicts containing image frame information.
        :param out_directory: A directory for the downloaded images.
        :return: True if the function succeeded; otherwise, False.
        """
        total_result = True
        for image_frame in image_frames:
            image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
            self.get_pixel_data(
                image_file_path,
                data_store_id,
                image_frame["imageSetId"],
                image_frame["imageFrameId"],
            )
```

```
            image_array = self.jph_image_to_opj_bitmap(image_file_path)
            crc32_checksum = image_frame["fullResolutionChecksum"]
            # Verify checksum.
            crc32_calculated = zlib.crc32(image_array)
            image_result = crc32_checksum == crc32_calculated
            print(
                f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
{image_result }"
            )
            total_result = total_result and image_result
        return total_result

    @staticmethod
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array
```

리소스를 정리합니다.

```
    def destroy(self, stack):
        """
        Destroys the resources managed by the CloudFormation stack, and the
CloudFormation
        stack itself.

        :param stack: The CloudFormation stack that manages the example
resources.
        """

        print(f"\t\tCleaning up resources and {stack.name}.")
```

```
        data_store_id = None
        for oput in stack.outputs:
            if oput["OutputKey"] == "DatastoreID":
                data_store_id = oput["OutputValue"]
        if data_store_id is not None:
            print(f"\t\tDeleting image sets in data store {data_store_id}.")
            image_sets = self.medical_imaging_wrapper.search_image_sets(
                data_store_id, {}
            )
            image_set_ids = [image_set["imageSetId"] for image_set in image_sets]

            for image_set_id in image_set_ids:
                self.medical_imaging_wrapper.delete_image_set(
                    data_store_id, image_set_id
                )
                print(f"\t\tDeleted image set with id : {image_set_id}")

        print(f"\t\tDeleting {stack.name}.")
        stack.delete()
        print("\t\tWaiting for stack removal. This may take a few minutes.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_delete_complete")
        waiter.wait(StackName=stack.name)
        print("\t\tStack delete complete.")




class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)
```

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

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


    def delete_image_set(self, datastore_id, image_set_id):
        """
        Delete an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        """
        try:
            delete_results = self.medical_imaging_client.delete_image_set(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't delete image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하세요.
  - DeleteImageSet
  - GetDICOMImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - StartDICOMImportJob

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 HealthImaging 데이터 스토어에 태그 지정

다음 코드 예제에서는 HealthImaging 데이터 스토어를 삭제하는 방법을 보여줍니다.

Java

SDK for Java 2.x

데이터 스토어에 태깅하려면.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
```

```
                                    ImmutableMap.of("Deployment", "Development"));
```

리소스에 태그를 지정하는 유틸리티 함수.

```
    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);
        }
    }
```

데이터 스토어의 태그를 나열하려면.

```
                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());
                }
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
    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;
    }
```

데이터 스토어에 태그 지정을 해제하려면.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

리소스의 태그를 해제하는 유틸리티 함수.

```
    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 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 항목을 참조하세요.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

데이터 스토어에 태깅하려면.

```
  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);
  }
```

리소스에 태그를 지정하는 유틸리티 함수.

```
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;
};
```

데이터 스토어의 태그를 나열하려면.

```
  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);
  }
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
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;
};
```

데이터 스토어에 태그 지정을 해제하려면.

```
  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);
  }
```

리소스의 태그를 해제하는 유틸리티 함수.

```
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 세부 정보는 AWS SDK for JavaScript API 참조의 다음 항목을 참조하세요.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

데이터 스토어에 태깅하려면.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

리소스에 태그를 지정하는 유틸리티 함수.

```
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
```

데이터 스토어의 태그를 나열하려면.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
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"]
```

데이터 스토어에 태그 지정을 해제하려면.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

리소스의 태그를 해제하는 유틸리티 함수.

```
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
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하십시오.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 HealthImaging 이미지 세트에 태그 지정

다음 코드 예제에서는 HealthImaging 이미지 세트에 태그를 지정하는 방법을 보여줍니다.

Java

SDK for Java 2.x

이미지 세트에 태그를 지정하려면.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

리소스에 태그를 지정하는 유틸리티 함수.

```
    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);
        }
    }
```

이미지 세트의 태그를 나열하려면.

```
                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());
                }
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
    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;
    }
```

이미지 세트의 태그를 해제하려면.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                Collections.singletonList("Deployment"));
```

리소스의 태그를 해제하는 유틸리티 함수.

```
    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 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 항목을 참조하세요.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

JavaScript

SDK for JavaScript (v3)

이미지 세트에 태그를 지정하려면.

```
  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);
  }
```

리소스에 태그를 지정하는 유틸리티 함수.

```
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;
};
```

이미지 세트의 태그를 나열하려면.

```
  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);
```

```
  }
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
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;
};
```

이미지 세트의 태그를 해제하려면.

```
  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);
  }
```

리소스의 태그를 해제하는 유틸리티 함수.

```
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 세부 정보는 AWS SDK for JavaScript API 참조의 다음 항목을 참조하세요.
  - ListTagsForResource

- TagResource
- UntagResource

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Python

SDK for Python(Boto3)

이미지 세트에 태그를 지정하려면.

```
    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"})
```

리소스에 태그를 지정하는 유틸리티 함수.

```
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
```

이미지 세트의 태그를 나열하려면.

```
    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)
```

리소스의 태그를 나열하는 유틸리티 함수입니다.

```
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"]
```

이미지 세트의 태그를 해제하려면.

```
    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"])
```

리소스의 태그를 해제하는 유틸리티 함수.

```
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
```

다음 코드는 MedicalImagingWrapper 객체를 인스턴스화합니다.

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

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하십시오.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서이 서비스 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS HealthImaging에서 DICOMweb 사용

의료 영상에 대한 DICOM 표준을 따르는 웹 기반 APIs인 DICOMweb APIs의 표현을 사용하여 AWS HealthImaging에서 DICOM 객체를 검색할 수 있습니다. 이 기능을 사용하면 HealthImaging의 클라우드 네이티브 작업을 활용하면서 DICOM Part 10 바이너리를 활용하는 시스템과 상호 운용할 수 있습니다. 이 장에서는 HealthImaging의 DICOMweb APIs 구현을 사용하여 DICOMweb 응답을 반환하는 방법을 중점적으로 다룹니다.

> **중요**
>
> HealthImaging은 DICOM 데이터를 이미지 세트로 저장합니다. HealthImaging 클라우드 네이티브 작업을 사용하여 이미지 세트를 관리하고 검색합니다. HealthImaging의 DICOMweb APIs 사용하여 DICOMweb 적합성 응답과 함께 이미지 세트 정보를 반환할 수 있습니다.
> 이 장에 나열된 APIs는 웹 기반 의료 영상에 대한 DICOMweb 표준을 준수하도록 구축되었습니다. DICOMweb APIs의 표현이므로 AWS CLI 및 AWS SDKs 통해 제공되지 않습니다.

주제

- HealthImaging에서 DICOM 데이터 검색
- HealthImaging에서 DICOM 데이터 검색

HealthImaging에서 DICOM 데이터 검색

AWS HealthImaging은 시리즈 및 인스턴스 수준에서 데이터를 검색하기 위한 `DICOMweb WADO-RS` APIs 표현을 제공합니다. 이러한 APIs 사용하면 HealthImaging 데이터 스토어에서 DICOM 시리즈의 모든 메타데이터를 검색할 수 있습니다. DICOM 인스턴스, DICOM 인스턴스 메타데이터 및 DICOM 인스턴스 프레임(픽셀 데이터)을 검색할 수도 있습니다. HealthImaging의 `DICOMweb WADO-RS` APIs HealthImaging에 저장된 데이터를 검색하고 레거시 애플리케이션과의 상호 운용성을 제공하는 방법에 유연성을 제공합니다.

> **중요**
>
> HealthImaging은 DICOM 데이터를 이미지 세트로 저장합니다. HealthImaging 클라우드 네이티브 작업을 사용하여 이미지 세트를 관리하고 검색합니다. HealthImaging의 DICOMweb APIs 사용하여 DICOMweb 적합성 응답과 함께 이미지 세트 정보를 반환할 수 있습니다.

이 섹션에 나열된 APIs는 웹 기반 의료 영상에 대한 DICOMweb(WADO-RS) 표준을 준수하도록 구축되었습니다. DICOMweb APIs의 표현이므로 AWS CLI 및 AWS SDKs 통해 제공되지 않습니다.

다음 표에서는 HealthImaging에서 데이터를 검색하는 데 사용할 수 있는 DICOMweb WADO-RS APIs의 모든 HealthImaging 표현을 설명합니다.

DICOMweb WADO-RS APIs의 HealthImaging 표현

명칭	설명
`GetDICOMSeriesMetadata`	리소스와 연결된 연구 및 시리즈 UIDs를 지정하여 HealthImaging 데이터 스토어에서 DICOM 시리즈에 대한 DICOM 인스턴스 메타데이터(`.json` 파일)를 검색합니다. 시리즈 메타데이터 가져오기을(를) 참조하세요.
`GetDICOMInstance`	리소스와 연결된 시리즈, 연구 및 인스턴스 UIDs를 지정하여 HealthImaging 데이터 스토어에서 DICOM 인스턴스(`.dcm` 파일)를 검색합니다. 인스턴스 가져오기을(를) 참조하세요.
`GetDICOMInstanceMetadata`	리소스와 연결된 시리즈, 연구 및 인스턴스 UIDs를 지정하여 HealthImaging 데이터 스토어의 DICOM 인스턴스에서 DICOM 인스턴스 메타데이터(`.json` 파일)를 검색합니다. 인스턴스 메타데이터 가져오기을(를) 참조하세요.
`GetDICOMInstanceFrames`	리소스와 연결된 시리즈 UID, 연구 UID, 인스턴스 UID UIDs 및 프레임 번호를 지정하여 HealthImaging 데이터 스토어의 DICOM 인스턴스에서 단일 또는 배치 이미지 프레임(`multipart` 요청)을 검색합니다. 인스턴스 프레임 가져오기을(를) 참조하세요.

주제

- HealthImaging에서 DICOM 인스턴스 가져오기

- HealthImaging에서 DICOM 인스턴스 메타데이터 가져오기
- HealthImaging에서 DICOM 인스턴스 프레임 가져오기
- HealthImaging에서 DICOM 시리즈 메타데이터 가져오기

HealthImaging에서 DICOM 인스턴스 가져오기

`GetDICOMInstance` 작업을 사용하여 리소스와 연결된 시리즈, 연구 및 인스턴스 UIDs를 지정하여 HealthImaging 데이터 스토어에서 DICOM 인스턴스(`.dcm` 파일)를 검색합니다. API는 선택적 이미지 세트 파라미터가 제공되지 않는 한 기본 이미지 세트의 인스턴스만 반환합니다. 를 쿼리 파라미터`imageSetId`로 지정하여 데이터 스토어의 모든 인스턴스(기본 또는 비기본 이미지 세트에서)를 검색할 수 있습니다. DICOM 데이터는 저장된 전송 구문 또는 비압축(ELE) 형식으로 검색할 수 있습니다.

DICOM 인스턴스를 가져오려면(**`.dcm`**)

1. HealthImaging `datastoreId` 및 `imageSetId` 파라미터 값을 수집합니다.
2. `GetImageSetMetadata` 작업을 `datastoreId` 및 `imageSetId` 파라미터 값과 함께 사용하여 `studyInstanceUID`, `seriesInstanceUID`및에 연결된 메타데이터 값을 검색합니다`sopInstanceUID`. 자세한 내용은 이미지 세트 메타데이터 가져오기 단원을 참조하십시오.
3. , `datastoreId`, `studyInstanceUID`, 및 값을 사용하여 요청에 대한 URL을 구성합니다`seriesInstanceUID``sopInstanceUID``imageSetId`. 다음 예제에서 전체 URL 경로를 보려면 복사 버튼을 스크롤합니다. URL의 형식은 다음과 같습니다.

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

4. 요청을 준비하고 전송합니다.는 AWS 서명 버전 4 서명 프로토콜과 함께 HTTP GET 요청을 `GetDICOMInstance` 사용합니다. 다음 코드 예제에서는 `curl` 명령줄 도구를 사용하여 HealthImaging에서 DICOM 인스턴스(`.dcm` 파일)를 가져옵니다.

   Shell

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

```
instances/1.3.6.1.4.1.5962.1.1.4.1.1.20040826186059.5457?
imageSetId=459e50687f121185f747b67bb60d1bc8' \
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom; transfer-syntax=1.2.840.10008.1.2.1' \
  --output 'dicom-instance.dcm'
```

Note

`transfer-syntax` UID는 선택 사항이며 포함되지 않은 경우 기본적으로 Explicit VR Little Endian으로 설정됩니다. 지원되는 전송 구문은 다음과 같습니다.

- 명시적 VR 리틀 엔디안(ELE) - `1.2.840.10008.1.2.1` (무손실 이미지 프레임의 기본값)
- RPCL 옵션 이미지 압축을 사용하는 High-Throughput JPEG 2000(Lossless Only) `1.2.840.10008.1.2.4.202` - 인스턴스가 HealthImaging에 로 저장된 경우 `1.2.840.10008.1.2.4.202`
- JPEG 기준(프로세스 1): 손실된 JPEG 8비트 이미지 압축에 대한 기본 전송 구문 `1.2.840.10008.1.2.4.50`- 인스턴스가 HealthImaging에 로 저장된 경우 `1.2.840.10008.1.2.4.50`
- JPEG 2000 이미지 압축 - 인스턴스가 HealthImaging에 로 저장된 `1.2.840.10008.1.2.4.91` 경우 `1.2.840.10008.1.2.4.91`
- High-Throughput JPEG 2000 Image Compression `1.2.840.10008.1.2.4.203` - 인스턴스가 HealthImaging에 로 저장된 경우 `1.2.840.10008.1.2.4.203`
- 전송 구문의 MPEG 패밀리(MPEG2, MPEG-4 AVC/H.264 및 HEVC/H.265 포함)에 인코딩된 하나 이상의 이미지 프레임이 있는 HealthImaging에 저장된 인스턴스는 해당 전송 구문 UID로 검색할 수 있습니다. 예를 들어 인스턴스가 MPEG2 기본 프로필 기본 레벨로 저장된 `1.2.840.10008.1.2.4.100` 경우입니다.

자세한 내용은 지원되는 전송 구문 및 AWS HealthImaging용 HTJ2K 디코딩 라이브러리 섹션을 참조하세요.

HealthImaging에서 DICOM 인스턴스 메타데이터 가져오기

`GetDICOMInstanceMetadata` 작업을 사용하여 리소스와 연결된 시리즈, 연구 및 인스턴스 UIDs를 지정하여 HealthImaging 데이터 스토어의 DICOM 인스턴스에서 메타데이터를 검색합니다. API는 선택적 이미지 세트 파라미터가 제공되지 않는 한 기본 이미지 세트의 인스턴스 메타데이터만 반환합니다. 를 쿼리 파라미터`imageSetId`로 지정하여 데이터 스토어에서 인스턴스 메타데이터(기본 또는 비기본 이미지 세트에서)를 검색할 수 있습니다.

DICOM 인스턴스 메타데이터를 가져오려면(**`.json`**)

1. HealthImaging `datastoreId` 및 `imageSetId` 파라미터 값을 수집합니다.
2. `GetImageSetMetadata` 작업을 `datastoreId` 및 `imageSetId` 파라미터 값과 함께 사용하여 `studyInstanceUID`, `seriesInstanceUID`및에 연결된 메타데이터 값을 검색합니다`sopInstanceUID`. 자세한 내용은 이미지 세트 메타데이터 가져오기 단원을 참조하십시오.
3. , `datastoreId`, `studyInstanceUID`, 및 값을 사용하여 요청에 대한 URL을 구성합니다`seriesInstanceUID``sopInstanceUID``imageSetId`. 다음 예제에서 전체 URL 경로를 보려면 복사 버튼을 스크롤합니다. URL의 형식은 다음과 같습니다.

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

4. 요청을 준비하고 전송합니다.는 AWS 서명 버전 4 서명 프로토콜과 함께 HTTP GET 요청을 `GetDICOMInstanceMetadata` 사용합니다. 다음 코드 예제에서는 `curl` 명령줄 도구를 사용하여 HealthImaging에서 DICOM 인스턴스 메타데이터(`.json` 파일)를 가져옵니다.

   Shell

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

> **Note**
> 메타데이터에 표시된 전송 구문 UID는 HealthImaging의 저장된 전송 구문 UID(`StoredTransferSyntaxUID`)와 일치합니다.

HealthImaging에서 DICOM 인스턴스 프레임 가져오기

`GetDICOMInstanceFrames` 작업을 사용하여 리소스와 연결된 시리즈 UID, 연구 UID, 인스턴스 UIDs 및 프레임 번호를 지정하여 HealthImaging 데이터 스토어의 DICOM 인스턴스에서 단일 또는 배치 이미지 프레임(`multipart` 요청)을 검색합니다. 이미지 세트 ID를 쿼리 파라미터로 제공하여 인스턴스 프레임을 검색할 이미지 세트를 지정할 수 있습니다. API는 선택적 이미지 세트 파라미터가 제공되지 않는 한 기본 이미지 세트의 인스턴스 프레임만 반환합니다. 를 쿼리 파라미터`imageSetId`로 지정하여 데이터 스토어의 모든 인스턴스 프레임(기본 또는 비기본 이미지 세트에서)을 검색할 수 있습니다.

DICOM 데이터는 저장된 전송 구문 또는 비압축(ELE) 형식으로 검색할 수 있습니다.

DICOM 인스턴스 프레임을 가져오려면(**`multipart`**)

1. HealthImaging `datastoreId` 및 `imageSetId` 파라미터 값을 수집합니다.
2. `GetImageSetMetadata` 작업을 `datastoreId` 및 `imageSetId` 파라미터 값과 함께 사용하여 `studyInstanceUID`, `seriesInstanceUID`및에 연결된 메타데이터 값을 검색합니다`sopInstanceUID`. 자세한 내용은 이미지 세트 메타데이터 가져오기 단원을 참조하십시오.
3. 연결된 메타데이터에서 검색할 이미지 프레임을 결정하여 `frameList` 파라미터를 형성합니다. `frameList` 파라미터는 임의의 순서로 하나 이상의 중복되지 않은 프레임 번호를 쉼표로 구분한 목록입니다. 예를 들어 메타데이터의 첫 번째 이미지 프레임은 프레임 1입니다.
   - 단일 프레임 요청: `/frames/1`
   - 다중 프레임 요청: `/frames/1,2,3,4`
4. , `datastoreId`, , `studyInstanceUID`, 및 값을 사용하여 요청에 대한 URL을 구성합니다`seriesInstanceUID``sopInstanceUID``imageSetId``frameList`. 다음 예제에서 전체 URL 경로를 보려면 복사 버튼을 스크롤합니다. URL의 형식은 다음과 같습니다.

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

5. 요청을 준비하고 전송합니다.는 AWS 서명 버전 4 서명 프로토콜과 함께 HTTP GET 요청을 `GetDICOMInstanceFrames` 사용합니다. 다음 코드 예제에서는 `curl` 명령줄 도구를 사용하여 HealthImaging의 `multipart` 응답으로 이미지 프레임을 가져옵니다.

   Shell

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

> **Note**
>
> `transfer-syntax` UID는 선택 사항이며 포함되지 않은 경우 기본적으로 Explicit VR Little Endian으로 설정됩니다. 지원되는 전송 구문은 다음과 같습니다.
>
> - 명시적 VR 리틀 엔디안(ELE) - `1.2.840.10008.1.2.1` (무손실 이미지 프레임의 기본값)
> - RPCL 옵션 이미지 압축을 사용하는 High-Throughput JPEG 2000(Lossless Only) `1.2.840.10008.1.2.4.202` - 인스턴스가 HealthImaging에 로 저장된 경우 `1.2.840.10008.1.2.4.202`
> - JPEG 기준(프로세스 1): 손실된 JPEG 8비트 이미지 압축에 대한 기본 전송 구문 `1.2.840.10008.1.2.4.50`- 인스턴스가 HealthImaging에 로 저장된 경우 `1.2.840.10008.1.2.4.50`
> - JPEG 2000 이미지 압축 - 인스턴스가 HealthImaging에 로 저장된 `1.2.840.10008.1.2.4.91` 경우 `1.2.840.10008.1.2.4.91`

- High-Throughput JPEG 2000 Image Compression `1.2.840.10008.1.2.4.203` - 인스턴스가 HealthImaging에 로 저장된 경우 `1.2.840.10008.1.2.4.203`
- 전송 구문의 MPEG 패밀리(MPEG2, MPEG-4 AVC/H.264 및 HEVC/H.265 포함)에 인코딩된 하나 이상의 이미지 프레임이 있는 HealthImaging에 저장된 인스턴스는 해당 전송 구문 UID로 검색할 수 있습니다. 예를 들어 인스턴스가 MPEG2 기본 프로필 기본 레벨로 저장된 `1.2.840.10008.1.2.4.100` 경우입니다.

자세한 내용은 지원되는 전송 구문 및 AWS HealthImaging용 HTJ2K 디코딩 라이브러리 섹션을 참조하세요.

HealthImaging에서 DICOM 시리즈 메타데이터 가져오기

`GetDICOMSeriesMetadata` 작업을 사용하여 HealthImaging 데이터 스토어에서 DICOM 시리즈(`.json` 파일)의 메타데이터를 검색합니다. 리소스와 연결된 연구 및 시리즈 UIDs를 지정하여 HealthImaging 데이터 스토어의 모든 기본 이미지 세트에 대한 시리즈 메타데이터를 검색할 수 있습니다. 이미지 세트 ID를 쿼리 파라미터로 제공하여 기본이 아닌 이미지 세트에 대한 시리즈 메타데이터를 검색할 수 있습니다. 시리즈 메타데이터는 `DICOM JSON` 형식으로 반환됩니다.

DICOM 시리즈 메타데이터를 가져오려면(**`.json`**)

1. HealthImaging `datastoreId` 및 `imageSetId` 파라미터 값을 수집합니다.
2. `datastoreId`, `studyInstanceUID`, `seriesInstanceUID`및 선택적으로 값을 사용하여 요청에 대한 URL을 구성합니다`imageSetId`. 다음 예제에서 전체 URL 경로를 보려면 복사 버튼을 스크롤합니다. URL의 형식은 다음과 같습니다.

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

3. 요청을 준비하고 전송합니다.는 AWS 서명 버전 4 서명 프로토콜과 함께 HTTP GET 요청을 `GetDICOMSeriesMetadata` 사용합니다. 다음 코드 예제에서는 `curl` 명령줄 도구를 사용하여 HealthImaging에서 메타데이터(`.json` 파일)를 가져옵니다.

   Shell

```
curl --request GET \
```

```
 'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
d9a2a515ab294163a2d2f4069eed584c/
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/metadata \
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom+json' \
  --output 'series-metadata.json'
```

선택적 `imageSetId` 파라미터를 사용합니다.

Shell

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

> **Note**
>
> - `imageSetId` 파라미터는 기본이 아닌 이미지 세트에 대한 시리즈 메타데이터를 검색하는 데 필요합니다. `GetDICOMInstanceMetadata` 작업은 `datastoreId`, `studyInstanceUID`가 지정된 경우에만 기본 이미지 세트에 대한 시리즈 메타데이터를 반환`seriesInstanceUID`합니다(제외`imagesetID`).

HealthImaging에서 DICOM 데이터 검색

AWS HealthImaging은 DICOMweb QIDO-RS APIs를 표현하여 환자 ID별로 연구, 시리즈 및 인스턴스를 검색하고 추가 사용을 위해 고유 식별자를 수신합니다. HealthImaging의 DICOMweb QIDO-RS

APIs HealthImaging에 저장된 데이터를 검색하고 레거시 애플리케이션과의 상호 운용성을 제공하는 방법에 유연성을 제공합니다.

> **중요**
>
> HealthImaging의 DICOMweb APIs 사용하여 QIDO-RS로 이미지 세트 정보를 반환할 수 있습니다. HealthImaging DICOMweb APIs 달리 명시되지 않는 한 이미지 세트만 참조합니다. HealthImaging 클라우드 네이티브 작업 또는 DICOMweb 작업의 선택적 이미지 세트 파라미터를 사용하여 기본이 아닌 이미지 세트를 검색합니다. HealthImaging의 DICOMweb APIs 사용하여 DICOMweb 적합성 응답과 함께 이미지 세트 정보를 반환할 수 있습니다.
> HealthImaging DICOMweb QIDO-RS 작업은 최대 10,000개의 레코드를 반환할 수 있습니다. 10,000개 이상의 리소스가 있는 경우 QIDO-RS 작업을 통해 검색할 수 없지만 DICOMweb WADO-RS 작업 또는 클라우드 네이티브 작업을 통해 검색할 수 있습니다.
> 이 섹션에 나열된 APIs는 웹 기반 의료 영상에 대한 DICOMweb(QIDO-RS) 표준을 준수하도록 구축되었습니다. AWS CLI 및 AWS SDKs 통해 제공되지 않습니다.

HealthImaging용 DICOMweb 검색 APIs

다음 표에서는 HealthImaging에서 데이터를 검색하는 데 사용할 수 있는 DICOMweb QIDO-RS APIs의 모든 HealthImaging 표현을 설명합니다.

DICOMweb QIDO-RS APIs의 HealthImaging 표현

명칭	설명
`SearchDICOMStudies`	GET 요청을 사용하여 검색 쿼리 요소를 지정하여 HealthImaging에서 DICOM 연구를 검색합니다. 연구 검색 결과는 마지막 업데이트, 날짜 내림차순(최신에서 가장 오래된) 순서로 JSON 형식으로 반환됩니다. 연구 검색을(를) 참조하세요.
`SearchDICOMSeries`	GET 요청을 사용하여 검색 쿼리 요소를 지정하여 HealthImaging에서 DICOM 시리즈를 검색합니다. 시리즈 검색 결과는 JSON 형식으로 반환되며 오름차순(가장 오래된 - 최신)`Series`

명칭	설명
	Number (0020, 0011)으로 정렬됩니다. 시리즈 검색을(를) 참조하세요.
SearchDICOMInstances	GET 요청을 사용하여 검색 쿼리 요소를 지정하여 HealthImaging에서 DICOM 인스턴스를 검색합니다. 인스턴스 검색 결과는 JSON 형식으로 반환되며, 오름차순(가장 오래된 - 최신)Instance Number (0020, 0013)으로 정렬됩니다. 인스턴스 검색을(를) 참조하세요.

HealthImaging에 지원되는 DICOMweb 쿼리 유형

HealthImaging은 연구, 시리즈 및 SOP 인스턴스 수준에서 QIDO-RS 계층적 리소스 쿼리를 지원합니다. HealthImaging에 QIDO-RS 계층 검색을 사용하는 경우:

- 연구를 검색하면 연구 목록이 반환됩니다.
- 연구 시리즈를 검색하려면 알려진가 필요하며 시리즈 목록을 StudyInstanceUID 반환합니다.
- 인스턴스 목록을 검색하려면 알려진 StudyInstanceUID 및가 필요합니다. SeriesInstanceUID

다음 표에서는 HealthImaging에서 데이터를 검색하는 데 지원되는 QIDO-RS 계층적 쿼리 유형을 설명합니다.

HealthImaging에서 지원하는 QIDO-RS 쿼리 유형

쿼리 유형	예제
속성 값 쿼리	가 있는 연구의 모든 시리즈를 검색합니다modality=CT . .../studies/1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894866/series?00080060=CT

쿼리 유형	예제
	환자 ID와 연구 날짜가 각각 이러한 값인 모든 연구를 검색합니다. `…/studies?PatientID=11235813&StudyDate=20130509`
키워드 쿼리	`SeriesInstanceUID` 키워드를 사용하여 모든 시리즈를 검색합니다. `.../studies/1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894866/series?SeriesInstanceUID=1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894868`
쿼리 태그 지정	그룹/요소 양식에 전달된 쿼리 파라미터를 사용하여 태그를 검색합니다. 0020000D와 같은 {group}{element}
범위 쿼리	`...?Modality=CT&StudyDate=AABBYYYY-BBCCYYYY`
`limit` 및를 사용한 결과 페이징 `offset`	`.../studies?limit=1&offset=0&00080020=20000101` 제한 및 오프셋 파라미터를 사용하여 검색 응답을 페이지 매김할 수 있습니다. 제한의 기본값은 1000이며 최대값은 섹션을 참조AWS HealthImaging 엔드포인트 및 할당량하세요. 최대 한도 = 1000, 최대 오프셋 = 9000

주제

- HealthImaging에서 DICOM 연구 검색

- HealthImaging에서 DICOM 시리즈 검색
- HealthImaging에서 DICOM 인스턴스 검색

HealthImaging에서 DICOM 연구 검색

`SearchDICOMStudies` API를 사용하여 HealthImaging 데이터 스토어에서 DICOM 연구를 검색합니다. 지원되는 DICOM 데이터 요소(속성)가 포함된 URL을 구성하여 HealthImaging에서 DICOM 연구를 검색할 수 있습니다. 연구 검색 결과는 마지막 업데이트, 날짜 내림차순(최신에서 가장 오래된) 순서로 JSON 형식으로 반환됩니다.

DICOM 연구를 검색하려면

1. HealthImaging `region` 및 `datastoreId` 값을 수집합니다. 자세한 내용은 데이터 스토어 속성 가져오기 단원을 참조하십시오.
2. 해당하는 모든 연구 요소를 포함하여 요청에 대한 URL을 구성합니다. 다음 예제에서 전체 URL 경로를 보려면 복사 버튼을 스크롤합니다. URL의 형식은 다음과 같습니다.

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

에 대한 연구 요소 **`SearchDICOMStudies`**

DICOM 요소 태그	DICOM 요소 이름
(0008,0020)	Study Date
(0008,0050)	Accession Number
(0008,0061)	Modalities in Study
(0008,0090)	Referring Physician Name
(0010,0010)	Patient Name
(0010,0020)	Patient ID
(0020,000D)	Study Instance UID
(0020,0010)	Study ID

3. 요청을 준비하고 전송합니다.는 AWS 서명 버전 4 서명 프로토콜과 함께 HTTP GET 요청을 `SearchDICOMStudies` 사용합니다. 다음 예제에서는 `curl` 명령줄 도구를 사용하여 DICOM 연구에 대한 정보를 검색합니다.

   curl

   ```
   curl --request GET \
     "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
   studies[?query]"
     --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
     --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
     --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
     --header 'Accept: application/dicom+json' \
     --output results.json
   ```

   연구 검색 결과는 마지막 업데이트, 날짜 내림차순(최신에서 가장 오래된) 순서로 JSON 형식으로 반환됩니다.

HealthImaging에서 DICOM 시리즈 검색

`SearchDICOMSeries` API를 사용하여 HealthImaging 데이터 스토어에서 DICOM 시리즈를 검색합니다. 지원되는 DICOM 데이터 요소(속성)가 포함된 URL을 구성하여 HealthImaging에서 DICOM 시리즈를 검색할 수 있습니다. 시리즈 검색 결과는 오름차순(가장 오래된~최신)으로 정렬된 JSON 형식으로 반환됩니다.

DICOM 시리즈를 검색하려면

1. HealthImaging `region` 및 `datastoreId` 값을 수집합니다. 자세한 내용은 데이터 스토어 속성 가져오기 단원을 참조하십시오.
2. `StudyInstanceUID` 값을 수집합니다. 자세한 내용은 이미지 세트 메타데이터 가져오기 단원을 참조하십시오.
3. 해당하는 모든 시리즈 요소를 포함하여 요청에 대한 URL을 구성합니다. 다음 예제에서 전체 URL 경로를 보려면 복사 버튼을 스크롤합니다. URL의 형식은 다음과 같습니다.

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

에 대한 시리즈 요소 **SearchDICOMSeries**

DICOM 요소 태그	DICOM 요소 이름
(0008,0060)	Modality
(0020,000E)	Series Instance UID

4. 요청을 준비하고 전송합니다.는 AWS 서명 버전 4 서명 프로토콜과 함께 HTTP GET 요청을 SearchDICOMSeries 사용합니다. 다음 예제에서는 curl 명령줄 도구를 사용하여 DICOM 시리즈 정보를 검색합니다.

   curl

   ```
   curl --request GET \
     "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
   studies/StudyInstanceUID/series[?query]"
     --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
     --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
     --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
     --header 'Accept: application/dicom+json' \
     --output results.json
   ```

   시리즈 검색 결과는 JSON 형식으로 반환되며, 오름차순(가장 오래된 - 최신)Series Number (0020,0011)으로 정렬됩니다.

HealthImaging에서 DICOM 인스턴스 검색

SearchDICOMInstances API를 사용하여 HealthImaging 데이터 스토어에서 DICOM 인스턴스를 검색합니다. 지원되는 DICOM 데이터 요소(속성)가 포함된 URL을 구성하여 HealthImaging에서 DICOM 인스턴스를 검색할 수 있습니다. 인스턴스 결과는 오름차순(가장 오래됨~최신)으로 정렬된 JSON 형식으로 반환됩니다.

DICOM 인스턴스를 검색하려면

1. HealthImaging region 및 datastoreId 값을 수집합니다. 자세한 내용은 데이터 스토어 속성 가져오기 단원을 참조하십시오.

2. `StudyInstanceUID` 및에 대한 값을 수집합니다`SeriesInstanceUID`. 자세한 내용은 이미지 세트 메타데이터 가져오기 단원을 참조하십시오.
3. 해당하는 모든 검색 요소를 포함하여 요청에 대한 URL을 구성합니다. 다음 예제에서 전체 URL 경로를 보려면 복사 버튼을 스크롤합니다. URL의 형식은 다음과 같습니다.

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

에 대한 인스턴스 요소 **`SearchDICOMInstances`**

DICOM 요소 태그	DICOM 요소 이름
`(0008,0016)`	`SOP Class UID`
`(0008,0018)`	`SOP Instance UID`

4. 요청을 준비하고 전송합니다.는 AWS 서명 버전 4 서명 프로토콜과 함께 HTTP GET 요청을 `SearchDICOMInstances` 사용합니다. 다음 예제에서는 `curl` 명령줄 도구를 사용하여 DICOM 인스턴스에 대한 정보를 검색합니다.

curl

```
curl --request GET \
  "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
studies/StudyInstanceUID/series/SeriesInstanceUID/instances[?query]"
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom+json' \
  --output results.json
```

인스턴스 검색 결과는를 기준으로 오름차순(가장 오래된 - 최신)`Instance Number` `(0020,0013)`으로 정렬된 JSON 형식으로 반환됩니다.

AWS HealthImaging 모니터링

모니터링 및 로깅은 AWS HealthImaging의 보안, 신뢰성, 가용성 및 성능을 유지하는 데 중요한 부분입니다.는 HealthImaging을 모니터링하고, 문제가 있을 때 보고하고, 적절한 경우 자동 조치를 취할 수 있는 다음과 같은 로깅 및 모니터링 도구를 AWS 제공합니다.

- AWS CloudTrail는 AWS 계정에서 또는 계정을 대신하여 수행한 API 호출 및 관련 이벤트를 캡처하고 사용자가 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 호출된 사용자 및 계정 AWS, 호출이 수행된 소스 IP 주소, 호출이 발생한 시기를 식별할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.
- Amazon CloudWatch는 AWS 리소스와 실행 중인 애플리케이션을 AWS 실시간으로 모니터링합니다. 지표를 수집 및 추적하고, 사용자 지정 대시보드를 생성할 수 있으며, 지정된 지표가 지정한 임곗값에 도달하면 사용자에게 알리거나 조치를 취하도록 경보를 설정할 수 있습니다. 예를 들어 CloudWatch에서 Amazon EC2 인스턴스의 CPU 사용량 또는 기타 지표를 추적하고 필요할 때 자동으로 새 인스턴스를 시작할 수 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.
- Amazon EventBridge: 애플리케이션을 다양한 소스의 데이터와 쉽게 연결할 수 있는 서버리스 이벤트 버스 서비스입니다. EventBridge는 자체 애플리케이션, 서비스형 소프트웨어(SaaS) 애플리케이션 및 AWS 서비스의 실시간 데이터 스트림을 제공한 다음, 해당 데이터를 Lambda 등의 대상으로 라우팅합니다. 이를 통해 서비스에서 발생하는 이벤트를 모니터링하고 이벤트 기반 아키텍처를 구축할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.

주제

- HealthImaging AWS CloudTrail 과 함께 사용
- HealthImaging에서 Amazon CloudWatch 사용
- HealthImaging에서 Amazon EventBridge 사용

HealthImaging AWS CloudTrail 과 함께 사용

AWS HealthImaging은 HealthImaging에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 HealthImaging에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 HealthImaging 콘솔의 호출과 HealthImaging API 작업에 대한 호출이 포함됩니다. 추적을 생성하면 HealthImaging 이벤트를 포함하여 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 전달할 수 있습니다. 트레일을 구성하지 않은 경우에도 CloudTrail

콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 사용자는 HealthImaging에 수행된 요청, 요청을 한 IP 주소, 요청을 한 사람, 요청을 한 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

추적 생성

계정을 생성할 AWS 계정 때에 대해 CloudTrail이 켜져 있습니다. HealthImaging에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 CloudTrail 이벤트 기록 보기를 참조하세요.

> **Note**
>
> 에서 AWS HealthImaging에 대한 CloudTrail 이벤트 기록을 보려면 속성 조회 메뉴로 AWS Management Console이동하여 이벤트 소스를 선택하고를 선택합니다`medical-imaging.amazonaws.com`.

HealthImaging 이벤트를 AWS 계정포함하여에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

> **Note**
>
> AWS HealthImaging은 관리 이벤트와 데이터 이벤트라는 두 가지 유형의 CloudTrail 이벤트를 지원합니다. 관리 이벤트는 HealthImaging을 포함하여 모든 AWS 서비스가 생성하는 일반적인 이벤트입니다. 기본적으로 로깅은 활성화된 모든 HealthImaging API 호출에 대한 관리 이벤트

에 적용됩니다. 데이터 이벤트는 요금이 청구되며 일반적으로 초당 트랜잭션(tps)이 많은 APIs 용으로 예약되므로 비용 목적으로 CloudTrail 로그를 사용하지 않도록 선택할 수 있습니다. HealthImaging을 사용하면 AWS HealthImaging API 참조에 나열된 모든 네이티브 API 작업이를 제외한 관리 이벤트로 분류됩니다`GetImageFrame`. `GetImageFrame` 작업은 CloudTrail을 데이터 이벤트로 사용하여 온보딩되므로 활성화해야 합니다. 자세한 내용을 알아보려면 AWS CloudTrail 사용 설명서의 데이터 이벤트 로깅을 참조하세요.
DICOMweb WADO-RS API 작업은 CloudTrail에서 데이터 이벤트로 분류되므로 옵트인해야 합니다. 자세한 내용은 사용 AWS CloudTrail 설명서의 HealthImaging에서 DICOM 데이터 검색 및 데이터 이벤트 로깅을 참조하세요.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에 의해 이루어졌는지 여부입니다.

자세한 내용은 CloudTrail `userIdentity` 요소를 참조하세요.

로그 항목 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 `GetDICOMImportJob` 작업을 보여주는 CloudTrail 로그 입력 항목을 보여줍니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "XXXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
        "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/
ce6d90ba-5fba-4456-a7bc-f9bc877597c3"
```

```
        "accountId": "123456789012",
        "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "XXXXXXXXXXXXXXXXXXXXX",
                "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
                "accountId": "123456789012",
                "userName": "TestAccessRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-28T15:52:42Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-28T16:02:30Z",
    "eventSource": "medical-imaging.amazonaws.com",
    "eventName": "GetDICOMImportJob",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-
Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync
 http/Apache cfg/retry-mode/standard",
    "requestParameters": {
        "jobId": "5d08d05d6aab2a27922d6260926077d4",
        "datastoreId": "12345678901234567890123456789012"
    },
    "responseElements": null,
    "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
    "eventID": "26307f73-07f4-4276-b379-d362aa303b22",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "824333766656",
    "eventCategory": "Management"
}
```


HealthImaging에서 Amazon CloudWatch 사용

원시 데이터를 수집해서 읽기 가능한 실시간에 가까운 지표로 처리하는 CloudWatch를 사용하여 AWS HealthImaging을 모니터링할 수 있습니다. 이러한 통계는 15개월간 보관되므로 기록 정보를 사용하여

웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 특정 임곗값을 주시하다가 해당 임곗값이 충족될 때 알림을 전송하거나 조치를 취하도록 경보를 설정할 수도 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.

> **Note**
>
> 지표는 모든 HealthImaging API에 대해 보고됩니다.

다음 표는 HealthImaging에서 사용 가능한 지표와 차원을 열거한 것입니다. 각각은 사용자가 지정한 데이터 범위의 빈도 수로 표시됩니다.

Metrics

Metrics	설명
호출 개수	API에 대한 호출 횟수. 이는 계정 또는 지정된 데이터 스토어에 대해 보고될 수 있습니다. 단위: 개 유효한 통계: Sum, Count 측정기준: 작업, 데이터 스토어 ID, 데이터 스토어 유형

AWS CLI, 또는 CloudWatch API를 사용하여 HealthImaging에 대한 지표 AWS Management Console를 가져올 수 있습니다. Amazon AWS 소프트웨어 개발 키트(SDK) 또는 CloudWatch API 도구 중 하나를 통해 CloudWatch API를 사용할 수 있습니다. HealthImaging 콘솔에는 CloudWatch API의 원시 데이터를 기반으로 그래프가 표시됩니다.

CloudWatch를 사용하여 HealthImaging을 모니터링하려면 적절한 CloudWatch 권한이 있어야 합니다. 자세한 내용은 CloudWatch 사용 설명서의 CloudWatch용 Identity 및 액세스 관리를 참조하세요.

HealthImaging 지표 보기

지표 보기(CloudWatch 콘솔)

1. 에 로그인 AWS Management Console 하고 CloudWatch 콘솔을 엽니다.
2. 지표, 모든 지표, AWS/Medical 이미징을 차례로 선택합니다.

3. 차원과 지표 이름을 선택한 다음 그래프에 추가를 선택합니다.
4. 날짜 범위 값을 선택합니다. 선택한 날짜 범위에 대한 지표 개수가 그래프에 표시됩니다.

CloudWatch를 사용하여 경보 생성

CloudWatch 경보는 지정한 기간 동안 단일 지표를 감시하고, Amazon Simple Notification Service (Amazon SNS) 주제 또는 Auto Scaling 정책에 알림 보내기와 같은 하나 이상의 작업을 수행합니다. 이러한 작업은 지정한 여러 기간 동안 지정된 임계값에 따른 지표의 값을 기반으로 합니다. 또한 CloudWatch는 경보로 인해 상태가 변경되면 Amazon SNS 메시지를 전송할 수 있습니다.

CloudWatch 경보는 상태가 변경되어 지정한 기간 동안 지속되는 경우에만 작업을 호출합니다. 자세한 내용은 CloudWatch 경보 사용을 참조하세요.

HealthImaging에서 Amazon EventBridge 사용

Amazon EventBridge는 이벤트를 사용하여 애플리케이션 구성 요소를 서로 연결하는 서버리스 서비스로, 확장 가능한 이벤트 기반 애플리케이션을 더 쉽게 구축할 수 있습니다. EventBridge의 기본은 이벤트를 대상으로 라우팅하는 규칙을 생성하는 것입니다. AWS HealthImaging은 상태 변경을 EventBridge에 지속적으로 전달합니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge란? 섹션을 참조하세요.

주제

- EventBridge로 전송된 HealthImaging 이벤트
- HealthImaging 이벤트 구조 및 예제

EventBridge로 전송된 HealthImaging 이벤트

다음 표에는 처리를 위해 EventBridge로 전송된 모든 HealthImaging 이벤트가 나열되어 있습니다.

HealthImaging 이벤트 유형	State
데이터 스토어 이벤트	
데이터 스토어 생성	`CREATING`
데이터 스토어 생성 실패	`CREATE_FAILED`

HealthImaging 이벤트 유형	State
데이터 스토어 생성됨	ACTIVE
데이터 스토어 삭제	DELETING
데이터 스토어 삭제됨	DELETED
자세한 내용은 AWS HealthImaging API 참조의 datastoreStatus를 참조하세요. HealthImaging	
작업 이벤트 가져오기	
가져오기 작업 제출됨	SUBMITTED
가져오기 작업 진행 중	IN_PROGRESS
가져오기 작업 완료	COMPLETED
가져오기 작업 실패	FAILED
자세한 내용은 AWS HealthImaging API 참조의 jobStatus를 참조하세요. HealthImaging	
이미지 세트 이벤트	
이미지 세트 생성됨	CREATED
이미지 세트 복사	COPYING
읽기 전용 액세스 권한이 있는 이미지 세트 복사	COPYING_WITH_READ_ONLY_ACCESS
이미지 세트 복사됨	COPIED
이미지 세트 복사 실패	COPY_FAILED
이미지 세트 업데이트	UPDATING
이미지 세트 업데이트됨	UPDATED
이미지 세트 업데이트 실패	UPDATE_FAILED
이미지 세트 삭제	DELETING

HealthImaging 이벤트 유형	State
이미지 세트 삭제됨	DELETED
자세한 내용은 AWS HealthImaging API 참조의 ImageSetWorkflowStatus를 참조하세요. HealthImaging	

HealthImaging 이벤트 구조 및 예제

HealthImaging 이벤트는 메타데이터 세부 정보도 포함하는 JSON 구조의 객체입니다. 메타데이터를 입력으로 사용하여 이벤트를 다시 생성하거나 자세한 내용을 알아볼 수 있습니다. 연결된 모든 메타데이터 필드는 다음 메뉴의 코드 예제 아래 테이블에 나열됩니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 이벤트 구조 참조를 참조하세요.

> **Note**
>
> HealthImaging 이벤트 구조의 `source` 속성은 입니다`aws.medical-imaging`.

데이터 스토어 이벤트

Data Store Creating

상태 - **`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

상태 - **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

상태 - **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

상태 - **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

상태 - **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"
    }
}
```

데이터 스토어 이벤트 - 메타데이터 설명

명칭	유형	설명
version	문자열	EventBridge 이벤트 스키마 버전입니다.
id	문자열	모든 이벤트에 대해 생성된 버전 4 UUID입니다.
detail-type	문자열	전송 중인 이벤트의 유형입니다.
source	문자열	이벤트를 생성한 서비스를 식별합니다.
account	문자열	데이터 스토어 소유자의 12자리 AWS 계정 ID입니다.
time	문자열	이벤트가 발생한 시간입니다.
region	문자열	데이터 스토어의 AWS 리전을 식별합니다.
resources	배열(문자열)	데이터 스토어의 ARN을 포함하는 JSON 배열입니다.
detail	객체	이벤트에 대한 정보를 포함하는 JSON 객체입니다.
detail.imagingVersion	문자열	HealthImaging의 이벤트 세부 정보 스키마에 대한 변경 사항을 추적하는 버전 ID입니다.

명칭	유형	설명
detail.datastoreId	문자열	상태 변경 이벤트와 연결된 데이터 스토어 ID입니다.
detail.datastoreName	문자열	데이터 스토어 이름입니다.
detail.datastoreStatus	문자열	현재 데이터 스토어 상태입니다.

작업 이벤트 가져오기

Import Job Submitted

상태 - **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

상태 - **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

상태 - **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

상태 - **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/"
    }
}
```

작업 이벤트 가져오기 - 메타데이터 설명

명칭	유형	설명
`version`	문자열	EventBridge 이벤트 스키마 버전입니다.
`id`	문자열	모든 이벤트에 대해 생성된 버전 4 UUID입니다.
`detail-type`	문자열	전송 중인 이벤트의 유형입니다.

명칭	유형	설명
source	문자열	이벤트를 생성한 서비스를 식별합니다.
account	문자열	데이터 스토어 소유자의 12자리 AWS 계정 ID입니다.
time	문자열	이벤트가 발생한 시간입니다.
region	문자열	데이터 스토어의 AWS 리전을 식별합니다.
resources	배열(문자열)	데이터 스토어의 ARN을 포함하는 JSON 배열입니다.
detail	객체	이벤트에 대한 정보를 포함하는 JSON 객체입니다.
detail.imagingVersion	문자열	HealthImaging의 이벤트 세부 정보 스키마에 대한 변경 사항을 추적하는 버전 ID입니다.
detail.datastoreId	문자열	상태 변경 이벤트를 생성한 데이터 스토어입니다.
detail.jobId	문자열	상태 변경 이벤트와 연결된 가져오기 작업 ID입니다.
detail.jobName	문자열	가져오기 작업 이름입니다.
detail.jobStatus	문자열	현재 작업 상태입니다.
detail.inputS3Uri	문자열	가져올 DICOM 파일이 포함된 S3 버킷의 입력 접두사 경로입니다.
detail.outputS3Uri	문자열	DICOM 가져오기 작업의 결과가 업로드될 S3 버킷의 출력 접두사입니다.

이미지 세트 이벤트

Image Set Created

상태 - **CREATED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Created",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "CREATED"
   }
}
```

Image Set Copying

상태 - **COPYING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
```

```
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING"
   }
}
```

Image Set Copying With Read Only Access

상태 - **COPYING_WITH_READ_ONLY_ACCESS**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying With Read Only Access",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS"
   }
}
```

Image Set Copied

상태 - **COPIED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copied",
   "source": "aws.medical-imaging",
```

```
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPIED"
   }
}
```

Image Set Copy Failed

상태 - **COPY_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copy Failed",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPY_FAILED"
   }
}
```

Image Set Updating

상태 - **UPDATING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updating",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "UPDATING"
   }
}
```

Image Set Updated

상태 - **UPDATED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updated",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATED"
```

```
   }
}
```

Image Set Update Failed

상태 - **UPDATE_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Update Failed",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATE_FAILED"
   }
}
```

Image Set Deleting

상태 - **DELETING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleting",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
```

```
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "DELETING"
   }
}
```

Image Set Deleted

상태 - **DELETED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleted",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "DELETED",
      "imageSetWorkflowStatus": "DELETED"
   }
}
```

이미지 세트 이벤트 - 메타데이터 설명

명칭	유형	설명
`version`	문자열	EventBridge 이벤트 스키마 버전입니다.

명칭	유형	설명
id	문자열	모든 이벤트에 대해 생성된 버전 4 UUID입니다.
detail-type	문자열	전송 중인 이벤트의 유형입니다.
source	문자열	이벤트를 생성한 서비스를 식별합니다.
account	문자열	데이터 스토어 소유자의 12자리 AWS 계정 ID입니다.
time	문자열	이벤트가 발생한 시간입니다.
region	문자열	데이터 스토어의 AWS 리전을 식별합니다.
resources	배열(문자열)	이미지 세트의 ARN을 포함하는 JSON 배열입니다.
detail	객체	이벤트에 대한 정보를 포함하는 JSON 객체입니다.
detail.imagingVersion	문자열	HealthImaging의 이벤트 세부 정보 스키마에 대한 변경 사항을 추적하는 버전 ID입니다.
detail.isPrimary	boolean	가져온 데이터가 관리형 계층 구조로 성공적으로 구성되었는지 또는 해결해야 할 메타데이터 충돌이 있는지 나타냅니다.
detail.imageSetVersion	문자열	인스턴스를 두 번 이상 가져오면 이미지 세트 버전이 증가합니다. 최신 버전은 기본 이미지 세트에 저장된 이전 버전을 덮어씁니다.

명칭	유형	설명
`detail.datastoreId`	문자열	상태 변경 이벤트를 생성한 데이터 스토어 ID입니다.
`detail.imagesetId`	문자열	상태 변경 이벤트와 연결된 이미지 세트 ID입니다.
`detail.imageSetState`	문자열	현재 이미지 세트 상태입니다.
`detail.imageSetWorkflowStatus`	문자열	현재 이미지 세트 워크플로 상태입니다.

의 보안 AWS HealthImaging

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다 AWS 클라우드. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. 에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 AWS 제공 범위 내 서비스규정 준수 프로그램 제공 범위 내 서비스를 AWS HealthImaging참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 HealthImaging 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 HealthImaging을 구성하는 방법을 보여줍니다. 또한 HealthImaging 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

주제

- AWS HealthImaging의 데이터 보호
- AWS HealthImaging 자격 증명 및 액세스 관리
- AWS HealthImaging의 규정 준수 검증
- AWS HealthImaging의 인프라 보안
- AWS CloudFormation을 사용하여 AWS HealthImaging 리소스 생성하기
- AWS HealthImaging 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)
- 에 대한 교차 계정 가져오기 AWS HealthImaging
- AWS HealthImaging의 복원성

AWS HealthImaging의 데이터 보호

AWS 공동 책임 모델 AWS HealthImaging의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 HealthImaging 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

주제

- 데이터 암호화
- 네트워크 트래픽 개인 정보 보호

데이터 암호화

AWS HealthImaging을 사용하면 확장성과 효율적인 암호화 기능을 제공하는 보안 계층을 클라우드의 저장 데이터에 추가하는 기능을 제공합니다. 다음이 포함됩니다.

- 대부분의 AWS 서비스에서 사용 가능한 저장 데이터 암호화 기능
- 암호화 키를 관리할지 아니면 자체 키를 완전히 제어할지 선택할 수 AWS Key Management Service 있는 유연한 키 AWS 관리 옵션입니다.
- AWS 소유 AWS KMS 암호화 키
- Amazon SQS의 서버 측 암호화(SSE)를 사용하여 민감한 데이터를 전송하는 암호화된 메시지 대기열.

또한는 암호화 및 데이터 보호를 AWS 환경에서 개발하거나 배포하는 모든 서비스와 통합할 수 있는 APIs를 AWS 제공합니다.

저장 시 암호화

기본적으로 HealthImaging은 서비스 소유 AWS Key Management Service 키를 사용하여 저장된 고객 데이터를 암호화합니다. 선택적으로 생성, 소유 및 관리하는 대칭 고객 관리형 AWS KMS 키를 사용하여 저장 데이터를 암호화하도록 HealthImaging을 구성할 수 있습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 대칭 암호화 KMS 키 생성을 참조하세요.

전송 중 암호화

HealthImaging은 TLS 1.2를 사용하여 퍼블릭 엔드포인트와 백엔드 서비스를 통해 전송 중 데이터를 암호화합니다.

키 관리

AWS KMS 키(KMS 키)는의 기본 리소스입니다 AWS Key Management Service. 외부에서 사용할 데이터 키를 생성할 수도 있습니다 AWS KMS.

AWS 소유 KMS 키

HealthImaging은 기본적으로 이러한 키를 사용하여 개인 식별 또는 개인 건강 정보(PHI) 저장 데이터와 같은 잠재적으로 민감한 정보를 자동으로 암호화합니다. AWS 소유 KMS 키는 계정에 저장되지 않습니다. 여러 AWS 계정에서 사용하기 위해를 AWS 소유하고 관리하는 KMS 키 모음의 일부입니다. AWS 서비스는 AWS 소유 KMS 키를 사용하여 데이터를 보호할 수 있습니다. AWS 소유 KMS 키를 확

인, 관리, 사용하거나 사용을 감사할 수 없습니다. 하지만 사용자는 데이터를 암호화하는 키를 보호하기 위해 어떤 작업을 수행하거나 어떤 프로그램을 변경할 필요가 없습니다.

AWS 소유 KMS 키를 사용하는 경우 월별 요금 또는 사용 요금이 부과되지 않으며 계정의 할당 AWS KMS 량에 포함되지 않습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서에서 AWS 소유 키를 참조하세요.

고객 관리형 KMS 키

AWS KMS 수명 주기 및 사용량을 완전히 제어하려는 경우 HealthImaging은 사용자가 생성, 소유 및 관리하는 대칭 고객 관리형 KMS 키 사용을 지원합니다. 사용자가 이 암호화 계층을 완전히 제어할 수 있으므로 다음과 같은 작업을 수행할 수 있습니다.

- 키 정책, IAM 정책 및 권한 부여 설정 및 유지 관리
- 키 암호화 자료 교체
- 키 정책 활성화 및 비활성화
- 태그 추가
- 키 별칭 만들기
- 삭제를 위한 스케줄 키

CloudTrail을 사용하여 HealthImaging이 사용자를 대신하여에 보내는 요청을 추적할 수도 AWS KMS 있습니다. 추가 AWS KMS 요금이 적용됩니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 고객 관리형 키를 참조하세요.

고객 관리형 키 생성하기

AWS Management Console 또는 AWS KMS APIs. 자세한 내용은 AWS Key Management Service 개발자 안내서의 대칭 암호화 KMS 키 생성을 참조하세요.

키 정책에서는 고객 관리형 키에 대한 액세스를 제어합니다. 모든 고객 관리형 키에는 키를 사용할 수 있는 사람과 키를 사용하는 방법을 결정하는 문장이 포함된 정확히 하나의 키 정책이 있어야 합니다. 고객 관리형 키를 만들 때 키 정책을 지정할 수 있습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 고객 관리형 키 액세스 관리를 참조하세요.

고객 관리형 키를 HealthImaging 리소스에서 사용하려면 키 정책에서 kms:CreateGrant 작업을 허용해야 합니다. 이렇게 하면 지정된 KMS 키에 대한 액세스를 제어하는 고객 관리형 키에 권한 부여가 추가되어 사용자에게 HealthImaging에 필요한 권한 부여 작업에 대한 액세스 권한이 부여됩니다. 자세한 내용은 AWS Key Management Service 개발자 가이드에서 AWS KMS권한 부여를 참조하세요.

고객 관리형 KMS 키를 HealthImaging 리소스에서 사용하려면 키 정책에서 다음 API 작업을 허용해야 합니다.

- `kms:DescribeKey`은 키를 검증하는 데 필요한 고객 관리형 키 세부 정보를 제공하는 합니다. 이것은 모든 작업에 필요합니다.
- `kms:GenerateDataKey`는 모든 쓰기 작업에 대해 저장된 리소스를 암호화할 수 있는 액세스를 제공합니다.
- `kms:Decrypt`는 암호화된 리소스의 읽기 또는 검색 작업에 대한 액세스를 제공합니다.
- `kms:ReEncrypt*`는 리소스를 재암호화할 수 있는 액세스를 제공합니다.

다음은 사용자가 해당 키로 암호화된 HealthImaging에서 데이터 스토어를 생성하고 상호 작용할 수 있도록 하는 정책 설명 예제입니다.

```
{
    "Sid": "Allow access to create data stores and perform CRUD and search in
 HealthImaging",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "medical-imaging.amazonaws.com"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyWithoutPlaintext"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:EncryptionContext:kms-arn": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f",
            "kms:EncryptionContext:aws:medical-imaging:datastoreId": "datastoreId"
        }
    }
}
```

고객 관리형 KMS 키를 사용할 때 필요한 IAM 권한

고객 관리형 KMS 키를 사용하여 AWS KMS 암호화가 활성화된 데이터 스토어를 생성하는 경우 HealthImaging 데이터 스토어를 생성하는 사용자 또는 역할에 대한 키 정책 및 IAM 정책 모두에 필요한 권한이 있습니다.

키 정책에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 IAM 정책 활성화를 참조하세요.

리포지토리를 생성하는 IAM 사용자, IAM 역할 또는 AWS 계정에는 아래 정책에 대한 권한과 AWS HealthImaging에 필요한 권한이 있어야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:GenerateDataKey",
                "kms:RetireGrant",
                "kms:Decrypt",
                "kms:ReEncrypt*",
            "Resource": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f"
        }
    ]
}
```

HealthImaging이에서 권한 부여를 사용하는 방법 AWS KMS

고객 관리형 KMS 키를 사용하려면 HealthImaging은 권한 부여가 필요합니다. 고객 관리형 KMS 키로 암호화된 데이터 스토어를 생성하면 HealthImaging은 CreateGrant 요청을에 전송하여 사용자를 대신하여 권한 부여를 생성합니다 AWS KMS. 의 권한 부여 AWS KMS 는 HealthImaging에 고객 계정의 KMS 키에 대한 액세스 권한을 부여하는 데 사용됩니다.

사용자를 대신하여 HealthImaging이 생성하는 권한 부여는 취소되거나 사용 중지 되지 않아야 합니다. 사용자 계정의 AWS KMS 키를 사용할 권한을 HealthImaging에 부여한 권한 부여를 취소하거나 사용 중지하면 HealthImaging은 이 데이터에 액세스하거나 데이터 스토어에 푸시된 새 이미지를 암호화하거나 보낼 때 복호화 할 수 없습니다. HealthImaging에 대한 권한 부여를 취소하거나 사용 중지하면 변

경 사항이 즉시 발생합니다. 액세스 권한을 취소하려면 권한 부여 취소 대신 데이터 스토어를 삭제합니다. 데이터 스토어가 삭제되면 HealthImaging은 사용자를 대신하여 부여 권한의 사용을 중지시킵니다.

HealthImaging을 위한 암호화 키 모니터링

CloudTrail을 사용하여 고객 관리형 KMS 키를 사용할 때 HealthImaging이 사용자를 대신하여 AWS KMS 에 보내는 요청을 추적할 수 있습니다. CloudTrail 로그의 로그 항목은 `userAgent` 필드의 `medical-imaging.amazonaws.com`에 나타나므로 HealthImaging에서 수행한 요청을 명확하게 구분할 수 있습니다.

다음 예제는 고객 관리형 키로 암호화된 데이터에 액세스`DescribeKey`하기 위해 HealthImaging에서 호출한 AWS KMS 작업을 모니터링하기 위한 `GenerateDataKey`, `CreateGrantDecrypt`, 및에 대한 CloudTrail 이벤트입니다.

다음은 `CreateGrant`를 이용해 HealthImaging이 고객이 제공한 KMS 키에 액세스하여, HealthImaging이 해당 KMS 키를 사용하여 저장된 모든 고객 데이터를 암호화할 수 있는 방법을 보여줍니다.

사용자가 자신의 권한 부여를 만들 필요는 없습니다. HealthImaging은 `CreateGrant` 요청을 전송하여 사용자를 대신하여 권한 부여를 생성합니다 AWS KMS. 의 권한 부여 AWS KMS 는 HealthImaging에 고객 계정의 AWS KMS 키에 대한 액세스 권한을 부여하는 데 사용됩니다.

```
{
            "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
            "GrantId":
 "44e88bc45b769499ce5ec4abd5ecb27eeb3b178a4782452aae65fe885ee5ba20",
            "Name": "MedicalImagingGrantForQIDO_ebff634a-2d16-4046-9238-e3dc4ab54d29",
            "CreationDate": "2025-04-17T20:12:49+00:00",
            "GranteePrincipal": "AWS Internal",
            "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "ReEncryptFrom",
                "ReEncryptTo",
                "CreateGrant",
                "RetireGrant",
                "DescribeKey"
```

```
            ]
        },
        {
            "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
            "GrantId":
 "9e5fd5ba7812daf75be4a86efb2b1920d6c0c9c0b19781549556bf2ff98953a1",
            "Name": "2025-04-17T20:12:38",
            "CreationDate": "2025-04-17T20:12:38+00:00",
            "GranteePrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "IssuingAccount": "AWS Internal",
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "ReEncryptFrom",
                "ReEncryptTo",
                "CreateGrant",
                "RetireGrant",
                "DescribeKey"
            ]
        },
        {
            "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
            "GrantId":
 "ab4a9b919f6ca8eb2bd08ee72475658ee76cfc639f721c9caaa3a148941bcd16",
            "Name": "9d060e5b5d4144a895e9b24901088ca5",
            "CreationDate": "2025-04-17T20:12:39+00:00",
            "GranteePrincipal": "AWS Internal",
            "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "ReEncryptFrom",
                "ReEncryptTo",
                "DescribeKey"
            ],
            "Constraints": {
```

```
                "EncryptionContextSubset": {
                    "kms-arn": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c"
                }
            }
        }
```

다음 예는 GenerateDataKey를 사용하여 데이터를 저장하기 전에 사용자에게 암호화하는 데 필요한 권한이 있는지 확인하는 방법을 보여줍니다.

```
{
    "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"
}
```

다음 예는 HealthImaging이 저장된 암호화된 데이터 키를 사용하여 암호화된 데이터에 액세스하는 `Decrypt` 작업을 호출하는 방법을 보여줍니다.

```
{
    "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"
}
```

다음 예제는 HealthImaging이 `DescribeKey` 작업을 사용하여 AWS KMS 고객 소유 AWS KMS 키가 사용 가능한 상태인지 확인하고 작동하지 않는 경우 사용자가 문제를 해결하는 데 도움이 되는 방법을 보여줍니다.

```
{
    "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"
}
```

자세히 알아보기

다음 리소스는 저장된 데이터 암호화에 대한 자세한 정보를 제공하며 AWS Key Management Service 개발자 안내서에 있습니다.

- AWS KMS 개념
- 에 대한 보안 모범 사례 AWS KMS

네트워크 트래픽 개인 정보 보호

HealthImaging과 온프레미스 애플리케이션 그리고 HealthImaging과 Amazon S3 사이의 트래픽이 보호됩니다. HealthImaging과 간의 트래픽은 기본적으로 HTTPS를 AWS Key Management Service 사용합니다.

- AWS HealthImaging은 미국 동부(버지니아 북부), 미국 서부(오레곤), 유럽(아일랜드) 및 아시아 태평양(시드니) 리전에서 사용할 수 있는 리전 서비스입니다.
- HealthImaging과 Amazon S3 버킷 사이의 트래픽은 전송 계층 보안(TLS)이 HealthImaging과 Amazon S3 사이, 그리고 HealthImaging과 이에 액세스하는 고객 애플리케이션 사이에 전송 중인 객체를 암호화 하므로 Amazon S3 버킷 IAM 정책에 `aws:SecureTransport condition`을 사용한 HTTPS(TLS) 암호화 연결만 허용해야 합니다. HealthImaging은 현재 퍼블릭 엔드포인트를 사용하여 Amazon S3 버킷의 데이터에 액세스하지만 이것이 데이터가 퍼블릭 인터넷을 통과한다는 의미는 아닙니다. HealthImaging과 Amazon S3 간의 모든 트래픽은 AWS 네트워크를 통해 라우팅되며 TLS를 사용하여 암호화됩니다.

AWS HealthImaging 자격 증명 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 누가 HealthImaging 리소스를 사용하도록 인증 받고(로그인 됨) 승인 되는지(권한을 가짐)를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리

- AWS HealthImaging에서 IAM을 사용하는 방법
- AWS HealthImaging에 대한 자격 증명 기반 정책 예제
- AWS AWS HealthImaging에 대한 관리형 정책
- HealthImaging의 교차 서비스 혼동된 대리자 방지
- AWS HealthImaging 자격 증명 및 액세스 문제 해결

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 HealthImaging에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 - HealthImaging 서비스를 사용하여 작업을 수행하는 경우, 관리자는 사용자가 필요한 보안 인증 및 권한을 제공합니다. 더 많은 HealthImaging 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. HealthImaging의 기능에 액세스할 수 없으면 AWS HealthImaging 자격 증명 및 액세스 문제 해결 섹션을 참조하십시오.

서비스 관리자 - 회사에서 귀하가 HealthImaging 리소스의 책임자라면 HealthImaging에 대한 완전한 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 HealthImaging 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사가 HealthImaging IAM을 사용하는 방법에 대해 자세히 알아보려면 AWS HealthImaging에서 IAM을 사용하는 방법 섹션을 참조하십시오.

IAM 관리자 - 귀하가 IAM 관리자라면 HealthImaging 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 HealthImaging 자격 증명 기반 정책 예제를 보려면 AWS HealthImaging에 대한 자격 증명 기반 정책 예제 섹션을 참조하십시오.

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. , AWS 계정 루트 사용자 IAM 사용자 또는 IAM 역할을 수임하여 인증(로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 자격 증명 AWS 으로에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의에 로그인하는 방법을 AWS참조하세요. AWS 계정

AWS 프로그래밍 방식으로에 액세스하는 경우는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어는 다중 인증(MFA)을 사용하여 계정의 보안을 강화할 것을 AWS 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 이 자격 증명을 AWS 계정 루트 사용자라고 하며 계정을 생성하는 데 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스 에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 사용자 디렉터리, 웹 자격 증명 공급자, AWS Directory Service, Identity Center 디렉터리 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스 에 액세스하는 모든 사용자의 사용자입니다. 페더레이션 자격 증명에 액세스할 때 역할을 AWS 계정수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 및 애플리케이션에서 사용할 수 있도록 자체 ID 소스의 사용자 AWS 계정 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격

증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 IAM 역할(콘솔)로 전환할 AWS Management Console수 있습니다. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 AWS CLI 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 정책을 리소스에 직접 연결할 AWS 서비스수 있습니다(역할을 프록시로 사용하는 대신). 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.

- 교차 서비스 액세스 - 일부는 다른에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 완료하기 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결된 AWS 경우 권한을 정의하는의 객체입니다.는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console AWS CLI, 또는 API에서 역할 정보를 가져올 수 있습니다 AWS .

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - SCPs는의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다 AWS Organizations. AWS Organizations 는 AWS 계정 기업이 소유한 여러을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각각을 포함하여 멤버 계정의 엔터티에 대한 권한을 제한합니다 AWS 계정 루트 사용자. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.
- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 AWS 계정 루트 사용자관계없이를 포함한 자격 증명에 대한 유효 권한에 영향을 미칠 수 있습니다. RCP를 AWS 서비스 지원하는 목록을 포함하여 조직 및 RCPs에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCPs)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

AWS HealthImaging에서 IAM을 사용하는 방법

IAM을 사용하여 HealthImaging에 대한 액세스를 관리하기 전에 HealthImaging에 사용할 수 있는 IAM 기능을 알아보십시오.

AWS HealthImaging에서 사용할 수 있는 IAM 기능

IAM 기능	HealthImaging 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	아니요
ABAC(정책 내 태그)	부분
임시 자격 증명	예
보안 주체 권한	예
서비스 역할	예
서비스 연결 역할	아니요

HealthImaging 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방식을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

HealthImaging에 대한 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지

를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

HealthImaging에 대한 자격 증명 기반 정책 예제

HealthImaging 자격 증명 기반 정책의 예를 보려면 AWS HealthImaging에 대한 자격 증명 기반 정책 예제 섹션을 참조하십시오.

HealthImaging의 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 다른 경우 신뢰할 수 AWS 계정있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에도 리소스에 액세스할 수 있는 권한을 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

HealthImaging에 대한 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

HealthImaging 작업 목록을 보려면 서비스 인증 참조의AWS HealthImaging 정의 작업을 참조하세요.

HealthImaging의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
AWS
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "AWS:action1",
      "AWS:action2"
         ]
```

HealthImaging 자격 증명 기반 정책의 예를 보려면 AWS HealthImaging에 대한 자격 증명 기반 정책 예제 섹션을 참조하십시오.

HealthImaging에 대한 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

HealthImaging 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 증명 참조의 AWS HealthImaging 정의 리소스를 참조하세요. 어떤 작업과 리소스를 ARN으로 사용할 수 있는지 알아보려면 AWS HealthImaging 정의 리소스를 참조하세요.

HealthImaging 자격 증명 기반 정책의 예를 보려면 AWS HealthImaging에 대한 자격 증명 기반 정책 예제 섹션을 참조하십시오.

HealthImaging에 사용되는 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition` 요소를 지정하거나 단일 `Condition` 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 `AND` 작업을 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우는 논리적 `OR` 작업을 사용하여 조건을 AWS 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS 는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

HealthImaging 조건 키 목록을 보려면 서비스 승인 참조의 AWS HealthImaging에 사용되는 조건 키를 참조하세요. 어떤 작업과 리소스를 조건 키로 사용할 수 있는지 알아보려면 AWS HealthImaging 정의 작업을 참조하세요.

HealthImaging 자격 증명 기반 정책의 예를 보려면 AWS HealthImaging에 대한 자격 증명 기반 정책 예제 섹션을 참조하십시오.

HealthImaging의 ACL

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

HealthImaging과 RBAC

RBAC 지원	예

IAM에 사용되는 기존 권한 부여 모델을 RBAC(역할 기반 액세스 제어)라고 합니다. RBAC는 AWS 외부에서 역할로 알려진 개인의 직무에 따라 권한을 정의합니다. 자세한 내용은 IAM 사용 설명서의 ABAC와 기존 RBAC 모델 비교를 참조하십시오.

HealthImaging과 ABAC

ABAC 지원(정책의 태그): 부분적

> **Warning**
>
> ABAC는 `SearchImageSets` API 작업을 통해 적용되지 않습니다. `SearchImageSets` 작업에 액세스할 수 있는 사람은 누구나 데이터 스토어의 이미지 세트에 대한 모든 메타데이터에 액세스할 수 있습니다.

> **Note**
>
> 이미지 세트는 데이터 스토어의 하위 리소스입니다. ABAC를 사용하려면 이미지 세트에 데이터 스토어와 동일한 태그가 있어야 합니다. 자세한 정보는 AWS HealthImaging으로 리소스에 태그 지정 섹션을 참조하십시오.

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. 에서는 AWS이러한 속성을 태그라고 합니다. IAM 엔터티(사용자 또는 역할) 및 많은 AWS 리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

HealthImaging에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

일부 AWS 서비스 는 임시 자격 증명을 사용하여 로그인할 때 작동하지 않습니다. 임시 자격 증명으로 AWS 서비스 작업하는를 비롯한 추가 정보는 AWS 서비스 IAM 사용 설명서의 IAM으로 작업하는 섹션을 참조하세요.

사용자 이름 및 암호를 제외한 방법을 AWS Management Console 사용하여에 로그인하는 경우 임시 자격 증명을 사용합니다. 예를 들어 회사의 SSO(Single Sign-On) 링크를 AWS 사용하여에 액세스하면 해당 프로세스가 임시 자격 증명을 자동으로 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다. 그런 다음 이러한 임시 자격 증명을 사용하여 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는 access AWS. AWS recommends에 액세스할 수 있습니다. 자세한 정보는 IAM의 임시 보안 자격 증명 섹션을 참조하세요.

HealthImaging의 교차 서비스 보안 주체 권한 부여

전달 액세스 세션(FAS) 지원: 예

IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 정책은 보안 주체에게 권한을 부여합니다. 일부 서비스를 사용할 때는 다른 서비스에서 다른 작업을 트리거하는 작업을 수행할 수 있습니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. 작업에 정책의 추가 종속 작업이 필요한지 여부를 확인하려면 서비스 권한 참조의 AWS HealthImaging 작업, 리소스 및 조건 키를 참조하세요.

HealthImaging 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
> 서비스 역할에 대한 권한을 변경하면 HealthImaging 기능이 중단될 수 있습니다. HealthImaging에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집하세요.

HealthImaging에 대한 서비스 연결 역할

서비스 링크 역할 지원: 아니요

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 `Yes`이(가) 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

AWS HealthImaging에 대한 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할은 HealthImaging 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARNs 형식을 포함하여 Awesome에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조의 AWS Awesome에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- HealthImaging 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

정책 모범 사례

ID 기반 정책이 HealthImaging 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부를 결정합니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

HealthImaging 콘솔 사용

AWS HealthImaging 콘솔에 액세스하려면 최소한의 권한 부여가 있어야 합니다. 이러한 권한은에서 HealthImaging 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 HealthImaging 콘솔을 계속 사용할 수 있도록 하려면 HealthImaging *ConsoleAccess* 또는 *ReadOnly* AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
```

```
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS AWS HealthImaging에 대한 관리형 정책

AWS 관리형 정책은에서 생성하고 관리하는 독립 실행형 정책입니다 AWS. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반적인 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다. 가 AWS 관리형 정책에 정의된 권한을 AWS 업데이트하면 정책이 연결된 모든 보안 주체 자격 증명(사용자, 그룹 및 역할)에 영향을 미칩니다. AWS 는 새 AWS 서비스 가 시작되거나 기존 서비스에 새 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

주제

- AWS 관리형 정책: AWSHealthImagingFullAccess
- AWS 관리형 정책: AWSHealthImagingReadOnlyAccess
- AWS 관리형 정책에 대한 HealthImaging 업데이트

AWS 관리형 정책: AWSHealthImagingFullAccess

`AWSHealthImagingFullAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 모든 HealthImaging 작업에 관리 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "medical-imaging:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "medical-imaging.amazonaws.com"
                }
            }
        }
    ]
}
```

AWS 관리형 정책: AWSHealthImagingReadOnlyAccess

`AWSHealthImagingReadOnlyAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 특정 AWS HealthImaging 에 대한 읽기 전용 권한을 부여합니다.

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

```
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "medical-imaging:GetDICOMImportJob",
            "medical-imaging:GetDatastore",
            "medical-imaging:GetImageFrame",
            "medical-imaging:GetImageSet",
            "medical-imaging:GetImageSetMetadata",
            "medical-imaging:ListDICOMImportJobs",
            "medical-imaging:ListDatastores",
            "medical-imaging:ListImageSetVersions",
            "medical-imaging:ListTagsForResource",
            "medical-imaging:SearchImageSets"
        ],
        "Resource": "*"
    }]
}
```

AWS 관리형 정책에 대한 HealthImaging 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 HealthImaging의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 참조 페이지의 RSS 피드를 구독하세요.

변경 사항	설명	날짜
HealthImaging이 변경 사항 추적 시작	HealthImaging은 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2023년 7월 19일

HealthImaging의 교차 서비스 혼동된 대리자 방지

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. AWS에서는 교차 서비스 가장으로 인해 혼동된 대리자 문제가 발생할 수 있습니다. 교차 서비스 가장은 한 서비스(호출하는 서비스)가 다른 서비스(호출되는 서비스)를 직접적으로 호출할 때 발생할 수 있습니다. 직접적으로 호출하는 서비스는 다른 고객의 리소스에 대

해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위해 AWS는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다.

AWS HealthImaging이 리소스에 다른 서비스를 제공하는 권한을 제한하려면 `ImportJobDataAccessRole` IAM 역할 신뢰 관계 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하는 것이 좋습니다. `aws:SourceArn`을 사용하면 하나의 리소스만 교차 서비스 액세스 권한과 연결됩니다. `aws:SourceAccount`를 사용하면 해당 계정의 모든 리소스가 교차 서비스 사용 권한과 연결됩니다. 두 글로벌 조건 컨텍스트 키를 모두 사용하는 경우 `aws:SourceAccount` 값과 `aws:SourceArn` 값에서 참조되는 계정은 동일한 정책 문에서 사용할 경우 동일한 계정 ID를 사용해야 합니다.

의 값은 영향을 받는 데이터 스토어의 ARN이어야 `aws:SourceArn` 합니다. 데이터 스토어의 전체 ARN을 모르거나 여러 데이터 스토어를 지정하는 경우 `aws:SourceArn` 글로벌 컨텍스트 조건 키를 ARN의 알 수 없는 부분에 * 와일드카드와 함께 사용합니다. 예를 들어, `aws:SourceArn~arn:aws:medical-imaging:us-west-2:111122223333:datastore/*`을 설정할 수 있습니다.

다음 신뢰 정책 예제에서는 `aws:SourceArn` 및 `aws:SourceAccount` 조건 키를 사용하여 데이터 스토어의 ARN을 기반으로 서비스 보안 주체에 대한 액세스를 제한하여 혼동된 대리자 문제를 방지합니다.

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

AWS HealthImaging 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 HealthImaging 및 IAM 사용 작업에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- HealthImaging에서 작업을 수행할 권한이 없습니다
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 HealthImaging 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

HealthImaging에서 작업을 수행할 권한이 없습니다

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `AWS:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 AWS:GetWidget on resource: my-example-widget
```

이 경우, `AWS:`*`GetWidget`* 작업을 사용하여 *`my-example-widget`* 리소스에 액세스할 수 있도록 `mateojackson` 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 사용자가 HealthImaging에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예시 오류는 `marymajor`(이)라는 IAM 사용자가 콘솔을 사용하여 HealthImaging에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 HealthImaging 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- HealthImaging이 이러한 기능을 지원하는지 여부를 알아보려면 AWS HealthImaging에서 IAM을 사용하는 방법 섹션을 참조하십시오.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 리소스에 대한 액세스 권한을 타사에 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

AWS HealthImaging의 규정 준수 검증

제3자 감사자는 여러 AWS 규정 준수 프로그램의 일환으로 AWS HealthImaging의 보안 및 규정 준수를 평가합니다. HealthImaging의 경우 여기에는 HIPAA가 포함됩니다.

특정 규정 준수 프로그램 범위의 AWS 서비스 목록은 규정 준수 프로그램 제공 범위 내 AWS 서비스규정 준수 프로그램 제공 . 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 에서 보고서 다운로드를 참조하세요 AWS Artifact.

AWS HealthImaging 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS 는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- AWS 파트너 솔루션 - 보안 및 규정 준수를 위한 자동 참조 배포 가이드는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수 중심 기준 환경을 배포하기 위한 단계를 제공합니다 AWS.
- HIPAA 보안 및 규정 준수 백서 설계 -이 백서에서는 기업이 AWS 를 사용하여 HIPAA 준수 애플리케이션을 생성하는 방법을 설명합니다.
- 의 GxP 시스템 AWS -이 백서는가 GxP 관련 규정 준수 및 보안에 AWS 접근하는 방법에 대한 정보를 제공하고 GxP의 맥락에서 AWS 서비스를 사용하는 방법에 대한 지침을 제공합니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 업계 및 위치에 적용될 수 있습니다.
- 규칙을 사용한 리소스 평가 - 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 AWS Config 평가합니다.
- AWS Security Hub -이 AWS 서비스는 보안 업계 표준 및 모범 사례 준수를 확인하는 데 도움이 AWS 되는 내 보안 상태에 대한 포괄적인 보기를 제공합니다.

AWS HealthImaging의 인프라 보안

관리형 서비스인 AWS HealthImaging은 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다. https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf

AWS 게시된 API 호출을 사용하여 네트워크를 통해 HealthImaging에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.3 이상을 지원해야 합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 PFS(전달 완전 보안, Perfect Forward Secrecy)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

AWS CloudFormation을 사용하여 AWS HealthImaging 리소스 생성하기

AWS HealthImaging은 AWS 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 리소스를 모델링하고 설정하는 데 도움이 되는 AWS CloudFormation서비스인와 통합됩니다. 원하

는 모든 AWS 리소스를 설명하고 해당 리소스를 AWS CloudFormation 프로비저닝하고 구성하는 템플릿을 생성합니다.

를 사용하면 템플릿을 재사용하여 HealthImaging 리소스를 일관되고 반복적으로 설정할 AWS CloudFormation수 있습니다. 리소스를 한 번 설명한 다음 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝합니다.

HealthImaging 및 AWS CloudFormation 템플릿

HealthImaging 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 AWS CloudFormation 템플릿을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이러한 템플릿은 AWS CloudFormation 스택에서 프로비저닝하려는 리소스를 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하여 AWS CloudFormation 템플릿을 시작할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS CloudFormation Designer이란 무엇입니까?를 참조하세요.

AWS HealthImaging은 AWS CloudFormation을 이용해 데이터 스토어 생성을 지원합니다. HealthImaging 데이터 스토어 프로비저닝에 대한 JSON 및 YAML 템플릿의 예를 비롯한 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS HealthImaging 리소스 유형 참조를 참조하십시오.

에 대해 자세히 알아보기 AWS CloudFormation

에 대해 자세히 알아보려면 다음 리소스를 AWS CloudFormation참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation API 레퍼런스
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

AWS HealthImaging 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 생성 AWS HealthImaging 하여 VPC와 간에 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 HealthImaging API에 프라이빗 액세스하기 위해 사용할 수 있는 기술인 AWS PrivateLink로 구동됩니다. VPC의 인스턴스는 HealthImaging API와 통신에 퍼블릭 IP 주소를 필요로 하지 않습니다. VPC와 HealthImaging 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다.

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

주제

- HealthImaging VPC 엔드포인트에 대한 고려 사항
- HealthImaging용 인터페이스 VPC 엔드포인트 생성
- HealthImaging용 VPC 엔드포인트 정책 생성

HealthImaging VPC 엔드포인트에 대한 고려 사항

HealthImaging에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한 사항을 검토해야 합니다.

HealthImaging은 VPC에서 모든 AWS HealthImaging 작업을 호출할 수 있도록 지원합니다.

HealthImaging용 인터페이스 VPC 엔드포인트 생성

Amazon VPC 콘솔이나 AWS Command Line Interface (AWS CLI)를 사용하여 HealthImaging 서비스에 대한 VPC 엔드포인트를 생성할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

다음 서비스 이름을 사용하여 HealthImaging용 VPC 엔드포인트를 생성합니다.

- com.amazonaws.*region*.medical-imaging
- com.amazonaws.*region*.runtime-medical-imaging
- com.amazonaws.*region*.dicom-medical-imaging

> **Note**
>
> PrivateLink를 사용하려면 프라이빗 DNS를 활성화해야 합니다.

리전의 기본 DNS 이름, 예를 들어 `medical-imaging.us-east-1.amazonaws.com`을 사용하여 HealthImaging에 API 요청을 보낼 수 있습니다.

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트를 통해 서비스 액세스를 참조하세요.

HealthImaging용 VPC 엔드포인트 정책 생성

HealthImaging에 대한 액세스를 제어하는 VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

예제: HealthImaging 작업용 VPC 엔드포인트 정책

다음은 HealthImaging용 엔드포인트 정책 예제입니다. 이 정책은 엔드포인트에 연결될 때 모든 리소스의 모든 보안 주체에게 HealthImaging에 액세스할 권한을 부여합니다.

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\":\"*\"}]}"
```

에 대한 교차 계정 가져오기 AWS HealthImaging

교차 계정/교차 리전 가져오기를 사용하면 지원되는 다른 리전에 있는 Amazon S3 버킷에서 HealthImaging 데이터 스토어로 데이터를 가져올 수 있습니다. ??? AWS 계정, 다른 AWS 조직에서 소유한 계정, 그리고 Open Data Registry of Open Data on에 있는 Imaging Data Commons(IDC)와 같은 열린 데이터 소스에서 데이터를 가져올 수 있습니다. AWS

HealthImaging 교차 계정/교차 리전 가져오기 사용 사례는 다음과 같습니다.

- 고객 계정에서 DICOM 데이터를 가져오는 Medical Imaging SaaS 제품
- 많은 Amazon S3 입력 버킷에서 하나의 HealthImaging 데이터 스토어를 채우는 대규모 조직
- 연구원이 다중 기관 임상 연구 간에 데이터를 안전하게 공유

교차 계정 가져오기를 사용하려면

1. Amazon S3 입력(소스) 버킷 소유자는 HealthImaging 데이터 스토어 소유자 `s3:ListBucket` 및 `s3:GetObject` 권한을 부여해야 합니다.
2. HealthImaging 데이터 스토어 소유자는 Amazon S3 버킷을 IAM에 추가해야 합니다`ImportJobDataAccessRole`. 가져오기용 IAM 역할 생성을 참조하세요.
3. HealthImaging 데이터 스토어 소유자는 가져오기 작업을 시작할 때 Amazon S3 입력 버킷에 `inputOwnerAccountId` 대해를 제공해야 합니다.

   > **Note**
   >
   > `inputOwnerAccountId`데이터 스토어 소유자는를 제공하여 Amazon S3 버킷이 지정된 계정에 속하는 입력을 검증하여 업계 표준 준수를 유지하고 잠재적 보안 위험을 완화합니다.

   다음 `startDICOMImportJob` 코드 예제에는 가져오기 작업 시작 섹션의 모든 AWS CLI 및 SDK 코드 예제에 적용할 수 있는 선택적 `inputOwnerAccountId` 파라미터가 포함되어 있습니다.

Java

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

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

    return "";
}
```

AWS HealthImaging의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다.는 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 AWS 리전 제공합니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

더 먼 지역적 거리를 두고 데이터 또는 애플리케이션을 복제해야 하는 경우 AWS 로컬 지역을 사용하세요. AWS 로컬 리전은 기존 AWS 리전을 보완하도록 설계된 단일 데이터 센터입니다. 모든 리전과 마찬가지로 AWS 리전 AWS 로컬 리전은 다른 리전과 완전히 격리됩니다 AWS 리전.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS HealthImaging 참조 자료

> **Note**
>
> 모든 네이티브 HealthImaging API 작업 및 데이터 형식은 AWS HealthImaging API 참조에 설명되어 있습니다.

주제

- AWS HealthImaging에 대한 DICOM 지원
- AWS HealthImaging 참조

AWS HealthImaging에 대한 DICOM 지원

AWS HealthImaging은 특정 DICOM 요소 및 전송 구문을 지원합니다. HealthImaging 메타데이터 키는 지원되는 환자, 연구 및 시리즈 수준의 DICOM 데이터 요소를 기반으로 하므로 이 요소들을 숙지하십시오. 가져오기를 시작하기 전에 의료 영상 데이터가 HealthImaging에서 지원하는 전송 구문 및 DICOM 요소 제약 조건을 준수하는지 확인하십시오.

> **Note**
>
> AWS HealthImaging은 현재 바이너리 분할 이미지 또는 아이콘 이미지 시퀀스 픽셀 데이터를 지원하지 않습니다.

주제

- 지원되는 SOP 클래스
- 메타데이터 정규화
- 지원되는 전송 구문
- DICOM 요소 제약 조건
- DICOM 메타데이터 제약 조건

지원되는 SOP 클래스

AWS HealthImaging을 사용하면 사용 중지 및 프라이빗을 포함하여 모든 SOP 클래스 UID로 인코딩된 DICOM P10 서비스 객체 페어(SOP) 인스턴스를 가져올 수 있습니다. 모든 프라이빗 속성도 보존됩니다.

메타데이터 정규화

DICOM P10 데이터를 AWS HealthImaging으로 가져오면 메타데이터와 이미지 프레임(픽셀 데이터)으로 구성된 이미지 세트로 변환됩니다. ??? 변환 프로세스 중에 특정 버전의 DICOM 표준을 기반으로 HealthImaging 메타데이터 키가 생성됩니다. HealthImaging은 현재 DICOM PS3.6 2022b 데이터 사전 기반의 메타데이터 키를 생성하고 지원합니다.

AWS HealthImaging은 환자, 연구 및 시리즈 수준에서 다음과 같은 DICOM 데이터 요소를 지원합니다.

환자 수준 요소

> **Note**
>
> 각 환자 수준 요소에 대한 자세한 설명은 DICOM 데이터 요소 레지스트리를 참조하십시오.

AWS HealthImaging은 다음과 같은 환자 수준 요소를 지원합니다.

```
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
```

연구 수준 요소

> **Note**
> 각 연구 수준 요소에 대한 자세한 설명은 DICOM 데이터 요소 레지스트리를 참조하십시오.

AWS HealthImaging은 다음과 같은 연구 수준 요소를 지원합니다.

```
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
```

시리즈 수준 요소

> **Note**
>
> 각 시리즈 수준 요소에 대한 자세한 설명은 DICOM 데이터 요소 레지스트리를 참조하십시오.

AWS HealthImaging은 다음과 같은 시리즈 수준 요소를 지원합니다.

```
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
```

지원되는 전송 구문

AWS HealthImaging은 전송 구문이 다른 DICOM P10 파일 가져오기를 지원합니다. 일부 파일은 가져오기 중에 원래 전송 구문 인코딩을 유지하는 반면, 다른 파일은 기본적으로 HTJ2K 무손실로 트랜스코딩됩니다. 다음 예제는 HealthImaging이에서 반환한 메타데이터 내에서 각 인스턴스에 `StoredTransferSyntaxUID` 대해를 기록하는 방법을 보여줍니다`GetImageSetMetadata`.

```
"Instances": {
   "999.999.2.19941105.134500.2.101": {
       "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.50",
       "ImageFrames": [{ ...
```

> **Note**
>
> 다음 표를 볼 때는 다음 사항에 유의하세요.
>
> - 별표(*)로 표시된 전송 구문 UID 항목은 파일이 가져오는 동안 원래 인코딩된 형식으로 저장되었음을 나타냅니다. 이러한 파일의 경우 인스턴스 메타데이터에 `StoredTransferSyntaxUID` 있는는 원래 전송 구문과 일치합니다.
> - 별표 없이 표시된 전송 구문 UID 항목은 파일을 가져오는 동안 파일이 HTJ2K 무손실로 트랜스코딩되어 HealthImaging에 저장되었음을 나타냅니다. 이러한 파일의 경우 인스턴스 메타데이터에 `StoredTransferSyntaxUID` 있는는 RPCL 옵션 이미지 압축을 사용하는 High-Throughput JPEG 2000—Lossless Only(1.2.840.10008.1.2.4.202)입니다.
> - `StoredTransferSyntaxUID` 키가 없거나 로 설정된 경우 `null`키가 HTJ2K 무손실 RPCL(1.2.840.10008.1.2.4.202)로 인코딩된다고 가정할 수 있습니다.

HealthImaging에서 지원하는 전송 구문

전송 구문 UID	전송 구문 이름
1.2.840.10008.1.2	암시적 VR Endian: DICOM의 기본 전송 구문

전송 구문 UID	전송 구문 이름
1.2.840.10008.1.2.1*(이진 분할은 원래 인코딩을 유지하는 반면, 비이진 분할은 HTJ2K 무손실 RPCL로 트랜스코딩됨)	명시적 VR Little Endian
1.2.840.10008.1.2.1.99	디플레이티드 명시적 VR Little Endian
1.2.840.10008.1.2.2	명시적 VR Big Endian
1.2.840.10008.1.2.4.50*	JPEG 베이스라인(프로세스 1): Lossy JPEG 8 비트 이미지 압축의 기본 전송 구문
1.2.840.10008.1.2.4.51	JPEG 베이스라인(프로세스 2 및 4): Lossy JPEG 12비트 이미지 압축의 기본 전송 구문(프로세스 4만 해당)
1.2.840.10008.1.2.4.57	JPEG 무손실 비계층적(프로세스 14)
1.2.840.10008.1.2.4.70	JPEG 무손실, 비계층, 1차 예측(프로세스 14 [선택한 값 1]): 무손실 JPEG 이미지 압축 기본 전송 구문
1.2.840.10008.1.2.4.80	JPEG-LS 무손실 이미지 압축
1.2.840.10008.1.2.4.81	JPEG-LS 손실(거의 무손실) 이미지 압축
1.2.840.10008.1.2.4.90	JPEG 2000 이미지 압축(무손실만)
1.2.840.10008.1.2.4.91*	JPEG 2000 이미지 압축
1.2.840.10008.1.2.4.201	고처리량 JPEG 2000 이미지 압축(무손실 전용)
1.2.840.10008.1.2.4.202	RPCL 옵션 이미지 압축을 사용하는 고처리량 JPEG 2000(무손실 전용)
1.2.840.10008.1.2.4.203*	처리량이 높은 JPEG 2000 이미지 압축
1.2.840.10008.1.2.5	RLE 무손실

전송 구문 UID	전송 구문 이름
1.2.840.10008.1.2.4.100*, 1.2.840.10008.1.2.4.100.1*	MPEG2 기본 프로필 기본 수준
1.2.840.10008.1.2.4.101*, 1.2.840.10008.1.2.4.101.1*	상위 수준의 MPEG2 기본 프로필
1.2.840.10008.1.2.4.102*, 1.2.840.10008.1.2.4.102.1*	MPEG-4 AVC/H.264 높은 프로필/레벨 4.1
1.2.840.10008.1.2.4.103*, 1.2.840.10008.1.2.4.103.1*	MPEG-4 AVC/H.264 BD 호환 High Profile/Level 4.1
1.2.840.10008.1.2.4.104*, 1.2.840.10008.1.2.4.104.1*	2D 비디오용 MPEG-4 AVC/H.264 High Profile/Level 4.2
1.2.840.10008.1.2.4.105*, 1.2.840.10008.1.2.4.105.1*	ITU-T H.264 비디오의 MPEG-4 AVC/H.264 High Profile/Level 4.2
1.2.840.10008.1.2.4.106*, 1.2.840.10008.1.2.4.106.1*	ITU-T H.264 비디오의 MPEG-4 AVC/H.264 스테레오 하이 프로파일/레벨 4.2
1.2.840.10008.1.2.4.107*	HEVC/H.265 기본 프로필/레벨 5.1 비디오
1.2.840.10008.1.2.4.108*	HEVC/H.265 기본 10 프로필/레벨 5.1

*가져오는 동안 원래 전송 구문 인코딩을 유지합니다.

DICOM 요소 제약 조건

의료 영상 데이터를 AWS HealthImaging으로 가져올 때는 다음 DICOM 요소에 최대 길이 제약이 적용됩니다. 가져오기에 성공하려면 데이터가 최대 길이 제한을 초과하지 않도록 해야 합니다.

가져오기 중 DICOM 요소 제약 조건

HealthImaging 키워드	DICOM 키워드	DICOM 키	길이 제한
DICOMPatientName	PatientName	(0010,0010)	최소: 0, 최대: 256

HealthImaging 키워드	DICOM 키워드	DICOM 키	길이 제한
DICOMPatientId	PatientID	(0010,0020)	최소: 0, 최대: 256
DICOMPatientBirthDate	PatientBirthDate	(0010,0030)	최소: 0, 최대: 18
DICOMPatientSex	PatientSex	(0010,0040)	최소: 0, 최대: 16
DICOMStudyInstanceUID	StudyInstanceUID	(0020,000D)	최소: 0, 최대: 256
DICOMStudyId	StudyID	(0020,0010)	최소: 0, 최대: 16
DICOMStudyDescription	StudyDescription	(0008,1030)	최소: 0, 최대: 64
DICOMNumberOfStudy RelatedSeries	NumberOfStudyRelatedSeries	(0020,1206)	최소: 0, 최대: 1,000,000
DICOMNumberOfStudy RelatedInstances	NumberOfStudyRelated 인스턴스	(0020,1208)	최소: 0, 최대: 10,000
DICOMAccessionNumber	AccessionNumber	(0008,0050)	최소: 0, 최대: 256
DICOMStudyDate	StudyDate	0008,0020	최소: 0, 최대: 18
DICOMStudyTime	StudyTime	(0008,0030)	최소: 0, 최대: 28

DICOM 메타데이터 제약 조건

`UpdateImageSetMetadata`를 사용하여 HealthImaging 메타데이터 속성을 업데이트하면 다음 DICOM 제약 조건이 적용됩니다.

- 업데이트 제약 조건이 `updatableAttributes` 및 모두에 적용되지 않는 한 Patient/Study/Series/인스턴스 수준 속성에서 프라이빗 속성을 업데이트하거나 제거할 수 없습니다. `removableAttributes`
- 다음과 같은 AWS HealthImaging 생성 속성을 업데이트할 수 없습니다: `SchemaVersion`, `DatastoreID`, `ImageSetID`, `PixelData`, `Checksum`, `Width`, `Height`, `MinPixelValue`, `MaxPixelValue`, `FrameSizeInBytes`
- `force` 플래그가 설정되지 않은 한 다음 DICOM 속성을 업데이트할 수 없습니다. `Tag.PixelData`, `Tag.StudyInstanceUID`, `Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID` `Tag.StudyID`
- `force` 플래그가 설정되지 않은 한 VR 유형 SQ(중첩 속성)로 속성을 업데이트할 수 없습니다.
- `force` 플래그가 설정되지 않으면 다중 값 속성을 업데이트할 수 없습니다.
- `force` 플래그가 설정되지 않은 한 속성 VR 유형과 호환되지 않는 값으로 속성을 업데이트할 수 없습니다.
- `force` 플래그가 설정되지 않은 한 DICOM 표준에 따라 유효한 속성으로 간주되지 않는 속성을 업데이트할 수 없습니다.
- 모듈 간에 속성을 업데이트할 수 없습니다. 예를 들어 고객 페이로드 요청에서 연구 수준에 환자 수준 속성이 제공된 경우 요청이 무효화될 수 있습니다.
- 관련 속성 모듈이 기존 `ImageSetMetadata`에 없는 경우 속성을 업데이트할 수 없습니다. 예를 들어, 기존 이미지 세트 메타데이터에 `seriesInstanceUID` 시리즈가 없는 경우 `seriesInstanceUID`에 대한 속성을 업데이트할 수 없습니다.

AWS HealthImaging 참조

이 섹션에는 AWS HealthImaging과 관련된 지원 데이터가 포함되어 있습니다.

주제

- AWS HealthImaging 엔드포인트 및 할당량
- AWS HealthImaging 제한 한계
- AWS HealthImaging 픽셀 데이터 확인
- AWS HealthImaging용 HTJ2K 디코딩 라이브러리
- AWS HealthImaging 샘플 프로젝트
- AWS SDK에서이 서비스 사용

AWS HealthImaging 엔드포인트 및 할당량

다음 주제에는 AWS HealthImaging 서비스 엔드포인트 및 할당량에 대한 정보가 포함되어 있습니다.

주제

- 서비스 엔드포인트
- Service quotas

서비스 엔드포인트

서비스 엔드포인트는 호스트 및 포트를 웹 서비스의 진입점으로 식별하는 URL입니다. 모든 웹 서비스 요청에는 진입점이 포함되어 있습니다. 대부분의 AWS 서비스는 특정 리전에 대한 엔드포인트를 제공하여 더 빠른 연결을 지원합니다. AWS HealthImaging에 대한 서비스 엔드포인트는 다음 테이블에 나열되어 있습니다.

리전 이름	지역	엔드포인트	프로토콜
미국 동부(버지니아 북부)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS
미국 서부(오레곤)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS
아시아 태평양(시드니)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS
유럽(아일랜드)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS

HTTP 요청을 사용하여 AWS HealthImaging 작업을 호출하는 경우 호출되는 작업에 따라 다른 엔드포인트를 사용해야 합니다. 다음 메뉴에는 HTTP 요청에 사용할 수 있는 서비스 엔드포인트와 해당 요청이 지원하는 작업이 나열되어 있습니다.

HTTP 요청에 대해 지원되는 API 작업

data store, import, tagging

엔드포인트를 통해 다음 데이터 스토어, 가져오기 및 태그 지정 작업에 액세스할 수 있습니다.

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- StartDICOMImportJob
- GetDICOMImportJob
- ListDICOMImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

엔드포인트를 통해 다음 이미지 세트 작업에 액세스할 수 있습니다.

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets

- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging은 DICOMweb Retrieve WADO-RS 서비스의 표현을 제공합니다. 자세한 내용은 HealthImaging에서 DICOM 데이터 검색 단원을 참조하십시오.

엔드포인트를 통해 다음 DICOMweb 서비스에 액세스할 수 있습니다.

```
https://dicom-medical-imaging.region.amazonaws.com
```

- `GetDICOMInstance`
- `GetDICOMInstanceMetadata`
- `GetDICOMInstanceFrames`

Service quotas

서비스 할당량은 AWS 계정의 리소스, 작업 및 항목에 대한 최대값으로 정의됩니다.

> **Note**
>
> 조정 가능한 할당량의 경우 Service Quotas 콘솔을 사용하여 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

AWS HealthImaging에 대한 기본 할당량은 다음 테이블에 나열되어 있습니다.

명칭	기본값	조정 가능	설명
데이터 스토어당 최대 동시 CopyImageSet 요청 수	지원되는 각 리전: 100	예	현재 AWS 리전의 데이터 스토어당 최대 동시 CopyImageSet 요청 수
데이터 스토어당 최대 동시 DeleteImageSet 요청 수	지원되는 각 리전: 100	예	현재 AWS 리전의 데이터 스토어당 최대 동시 DeleteImageSet 요청 수
데이터 스토어당 최대 동시 UpdateImageSetMetadata 요청 수	지원되는 각 리전: 100	예	현재 AWS 리전의 데이터 스토어당 최대 동시 UpdateImageSetMetadata 요청 수
데이터 스토어당 최대 동시 가져오기 작업 수	ap-southeast-2: 20 각각의 지원되는 다른 리전: 100	예	현재 AWS 리전의 데이터 스토어당 최대 동시 가져오기 작업 수
최대 데이터 스토어	지원되는 각 리전: 10	예	현재 AWS 리전의 최대 활성 데이터 스토어 수
CopyImageSet 요청당 복사할 수 있는 최대 ImageFrames 수	지원되는 각 리전: 1,000개	예	현재 AWS 리전에서 CopyImageSet 요청 당 복사할 수 있는 최대 ImageFrames 수
DICOM 가져오기 작업의 최대 파일 수	각 지원되는 리전: 5,000	예	현재 AWS 리전의 DICOM 가져오기 작업에서 최대 파일 수

명칭	기본값	조정 가능	설명
DICOM 가져오기 작업의 최대 중첩 폴더 수	지원되는 각 리전: 10,000개	아니요	현재 AWS 리전의 DICOM 가져오기 작업에서 중첩된 폴더의 최대 수
UpdateImageSetMetadata에서 허용하는 최대 페이로드 크기 한도(KB)	지원되는 각 리전: 10킬로바이트	예	현재 AWS 리전에서 UpdateImageSetMetadata가 수락한 최대 페이로드 크기 제한(KB)
DICOM 가져오기 작업에 있는 모든 파일의 최대 크기(GB)	지원되는 각 리전: 10기가바이트	아니요	현재 AWS 리전의 DICOM 가져오기 작업에 있는 모든 파일의 최대 크기(GB)
DICOM 가져오기 작업에 있는 각 DICOM P10의 최대 크기(GB)	지원되는 각 리전: 4기가바이트	아니요	현재 AWS 리전의 DICOM 가져오기 작업에서 각 DICOM P10 파일의 최대 크기(GB)
가져오기, 복사 및 UpdateImageSet당 ImageSetMetadata의 최대 크기 제한(MB)	지원되는 각 리전: 50메가바이트	예	현재 AWS 리전의 가져오기, 복사 및 UpdateImageSet당 ImageSetMetadata의 최대 크기 제한(MB)

AWS HealthImaging 제한 한계

AWS 계정에 AWS HealthImaging API 작업에 적용되는 제한 한도가 있습니다. 모든 작업에 대해 제한 한계를 초과하면 `ThrottlingException` 오류가 발생합니다. 자세한 내용은 AWS HealthImaging API Reference를 참조하십시오.

> **Note**
>
> 모든 HealthImaging API 작업에 대해 제한 한도를 조정할 수 있습니다. 제한 한도 조정을 요청하려면 AWS Support Center에 문의하세요. 사례를 생성하려면 AWS 계정에 로그인하고 사례 생성을 선택합니다.

다음 표에는 DICOMweb 서비스의 기본 HealthImaging 작업 및 표현에 대한 제한 제한이 나와 있습니다. DICOMweb

AWS HealthImaging 제한 한계

작업	제한 속도	제한 버스트
CreateDatastore	0.085 tps	1 TPS
GetDatastore	10 tps	20 tps
ListDatastores	5 tps	10 tps
DeleteDatastore	0.085 tps	1 tps
StartDICOMImportJob	0.25 tps	1 tps
GetDICOMImportJob	25 tps	50 tps
ListDICOMImportJobs	10 tps	20 tps
SearchImageSets	25 tps	50 tps
GetImageSet	25 tps	50 tps
GetImageSetMetadata	50 tps	100 tps
GetImageFrame	1000 tps	2000 tps
ListImageSetVersions	25 tps	50 tps
UpdateImageSetMetadata	0.25 tps	1 tps
CopyImageSet	0.25 tps	1 tps

작업	제한 속도	제한 버스트
DeleteImageSet	0.25 tps	1 tps
TagResource	10 tps	20 tps
ListTagsForResource	10 tps	20 tps
UntagResource	10 tps	20 tps
GetDICOMInstance*	50 tps	100 tps
GetDICOMInstanceMetadata*	50 tps	100 tps
GetDICOMInstanceFrames*	50 tps	100 tps
GetDICOMSeriesMetadata	50 tps	100 tps

*DICOMweb 서비스 표시

AWS HealthImaging 픽셀 데이터 확인

가져오는 동안 HealthImaging은 모든 이미지의 무손실 인코딩 및 디코딩 상태를 확인하여 기본 제공 픽셀 데이터 확인을 제공합니다. 이 기능은 HTJ2K 디코딩 라이브러리를 사용하여 디코딩된 이미지가 항상 HealthImaging으로 가져온 원래 DICOM P10 이미지와 일치하도록 합니다.

- 이미지 온보딩 프로세스는 가져오기 작업이 DICOM P10 이미지를 가져오기 전에 DICOM P10 이미지의 원래 픽셀 품질 상태를 캡처할 때 시작됩니다. CRC32 알고리즘을 사용하여 각 이미지에 대해 고유한 변경 불가능한 이미지 프레임 해상도 체크섬(IFRC)이 생성됩니다. IFRC 체크섬 값은 `job-output-manifest.json` 메타데이터 문서에 표시됩니다. 자세한 내용은 가져오기 작업에 대한 이해 단원을 참조하십시오.
- 이미지를 HealthImaging 데이터 스토어로 가져오고 이미지 세트로 변환하면 HTJ2K-encoded 이미지 프레임이 즉시 디코딩되고 새 IFRCs가 계산됩니다. 그런 다음 HealthImaging은 원본 이미지의 전체 해상도 IFRCs를 가져온 이미지 프레임의 새 IFRCs와 비교하여 정확도를 확인합니다.
- 해당 이미지별 설명 오류 조건은 검토 및 확인할 수 있도록 가져오기 작업 출력 로그(`job-output-manifest.json`)에 캡처됩니다.

픽셀 데이터를 확인하려면

1. 의료 영상 데이터를 가져온 후, `job-output-manifest.json`의 가져오기 작업 출력 로그에 캡처된 이미지 세트별 설명 성공(또는 오류 조건)을 확인하세요. 자세한 내용은 가져오기 작업에 대한 이해 단원을 참조하십시오.
2. 이미지 세트는 메타데이터와 이미지 프레임(픽셀 데이터)으로 구성됩니다. 이미지 세트 메타데이터에는 연결된 이미지 프레임에 대한 정보가 포함되어 있습니다. `GetImageSetMetadata` 작업을 사용하여 이미지 세트에 대한 메타데이터를 가져옵니다. 자세한 내용은 이미지 세트 메타데이터 가져오기 단원을 참조하십시오.
3. 에는 전체 해상도 이미지에 대한 IFRC(체크섬)가 `PixelDataChecksumFromBaseToFullResolution` 포함되어 있습니다. 원래 전송 구문 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 및 1.2.840.10008.1.2.1(이진 분할만 해당)에 저장된 이미지의 경우 원본 이미지에서 체크섬이 계산됩니다. RPCL을 사용하여 HTJ2K Lossless에 저장된 이미지의 경우 디코딩된 전체 해상도 이미지에서 체크섬이 계산됩니다. 자세한 내용은 지원되는 전송 구문 단원을 참조하십시오.

   다음은 가져오기 작업 프로세스의 일부로 생성되어에 기록되는 IFRC에 대한 메타데이터 출력의 예입니다`job-output-manifest.json`.

   ```
   "ImageFrames": [{
   "ID": "6789067890678901234512345123451234512",
   "PixelDataChecksumFromBaseToFullResolution": [
   {
       "Width": 512,
       "Height": 512,
       "Checksum": 2510355201
   }
   ]
   ```

   원래 전송 구문 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 및 1.2.840.10008.1.2.1(이진 분할만 해당)에 저장된 이미지`MaxPixelValue`의 경우 `MinPixelValue` 및를 사용할 수 없습니다. 는 원래 프레임의 크기를 `FrameSizeInBytes` 나타냅니다.

   ```
   "PixelDataChecksumFromBaseToFullResolution": [
      {"Width": 512, "Height": 512, "Checksum": 1379921327 }
   ],
   "MinPixelValue": null,
   "MaxPixelValue": null,
   ```

```
"FrameSizeInBytes": 429
```

RPCL을 사용하여 HTJ2K Lossless에 저장된 이미지의 경우는 디코딩된 이미지 프레임의 크기를 `FrameSizeInBytes` 나타냅니다.

```
"PixelDataChecksumFromBaseToFullResolution": [
   {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": 11,
"MaxPixelValue": 11,
"FrameSizeInBytes": 1652
```

4. 비디오가 포함된 인스턴스의 경우 HealthImaging은 DICOM 메타데이터에 지정된 전송 구문이 비디오 코덱과 일치하는지 확인하는 경량 코덱 검증을 수행합니다.

   HealthImaging은 CRC32 알고리즘을 사용하여 원본 비디오 객체에 대한 IFRC 체크섬 값을 계산합니다. IFRC 체크섬 값은에 기록`job-output-manifest.json`되며 HealthImaging 메타데이터에 유지됩니다. 원본 전송 구문(위에 설명됨)에 저장된 이미지와 마찬가지로 `MinPixelValue` 및 `MaxPixelValue`는 사용할 수 없습니다. FrameSizeInBytes는 원래 프레임의 크기를 나타냅니다.

5. HealthImagingHTJ2K 디코딩 라이브러리은 픽셀 데이터를 확인하고, GitHub의 Pixel 데이터 확인 절차에 액세스하고, 파일의 지침에 따라 HealthImaging에서`README.md` 사용하는 다양한에서 손실 없는 이미지 처리를 독립적으로 확인합니다. 전체 이미지가 로드된 후 엔드에서 원시 입력 데이터의 IFRC를 계산하고 HealthImaging 메타데이터에 제공된 IFRC 값과 비교하여 픽셀 데이터를 확인할 수 있습니다.

AWS HealthImaging용 HTJ2K 디코딩 라이브러리

가져오기 중에 일부 전송 구문은 원래 인코딩을 유지하고 다른 전송 구문은 기본적으로 High-Throughput JPEG 2000(HTJ2K) 무손실로 트랜스코딩됩니다. HTJ2K는 일관되게 빠른 이미지 표시와 HTJ2K의 고급 기능에 대한 범용 액세스를 제공합니다. 일부 이미지 프레임은 가져오는 동안 HTJ2K로 인코딩되므로 이미지 뷰어에서 보기 전에 디코딩해야 합니다. 전송 구문 결정에 대한 자세한 내용은 섹션을 참조하세요지원되는 전송 구문.

> **Note**
>
> HTJ2K는 JPEG2000 표준(ISO/IEC 15444-15:2019)의 파트 15에 정의되어 있습니다. HTJ2K는 해상도 확장성, 구역, 타일링, 높은 비트 심도, 다중 채널 및 색 공간 지원과 같은 JPEG2000 고급 기능을 유지합니다.

주제

- HTJ2K 디코딩 라이브러리
- 이미지 뷰어

HTJ2K 디코딩 라이브러리

프로그래밍 언어에 따라 이미지 프레임을 디코딩하려면 다음 디코딩 라이브러리를 사용하는 것이 좋습니다.

- NVIDIA nvJPEG2000 - 상용, GPU 가속화
- Kakadu 소프트웨어 - 상용, C++ (Java 및 .NET 바인딩 포함)
- OpenJPH - 오픈 소스, C++ 및 WASM
- OpenJPH - 오픈 소스, C/C++, Java
- openjphpy - 오픈 소스, Python
- pylibjpeg-openjpeg - 오픈 소스, Python

이미지 뷰어

이미지 프레임을 디코딩한 후 볼 수 있습니다. AWS HealthImaging API 작업은 다음을 포함한 다양한 오픈 소스 이미지 뷰어를 지원합니다.

- Open Health Imaging Foundation(OHIF)
- Cornerstone.js

AWS HealthImaging 샘플 프로젝트

AWS HealthImaging은 GitHub에서 다음과 같은 샘플 프로젝트를 제공합니다.

온프레미스에서 AWS HealthImaging까지 DICOM 수집

DICOM DIMSE 소스(PACS, VNA, CT 스캐너)에서 DICOM 파일을 수신하여 안전한 Amazon S3 버킷에 저장하는 IoT 엣지 솔루션을 배포하기 위한 AWS 서버리스 프로젝트입니다. 이 솔루션은 데이터베이스의 DICOM 파일을 인덱싱하고 AWS HealthImaging으로 가져올 각 DICOM 시리즈를 대기열에 넣습니다. 에서 관리하는 엣지에서 실행되는 구성 요소와 AWS 클라우드에서 실행되는 DICOM 수집 파이프라인AWS IoT Greengrass으로 구성됩니다.

타일 레벨 마커(TLM) 프록시

처리량이 많은 JPEG 2000(HTJ2K)의 기능인 타일 레벨 마커(TLM)를 사용하여 AWS HealthImaging에서 이미지 프레임을 검색하는 AWS 클라우드 개발 키트 (AWS CDK) 프로젝트입니다. 그 결과 해상도가 낮은 이미지를 더 빠르게 검색할 수 있습니다. 가능한 워크플로우에는 썸네일 생성 및 점진적 이미지 로딩이 포함됩니다.

Amazon CloudFront 전송

(GET을 사용하여) 캐시하고 엣지에서 이미지 프레임을 제공하는 HTTPS 엔드포인트를 사용하여 Amazon CloudFront 배포를 생성하기 위한 AWS 서버리스 프로젝트입니다. 기본적으로 엔드포인트는 Amazon Cognito JSON 웹 토큰(JWT)을 사용하여 요청을 인증합니다. 인증과 요청 서명 모두 Lambda @Edge를 사용하여 엣지에서 수행됩니다. 이 서비스는 Amazon CloudFront의 기능으로, 애플리케이션 사용자와 더 가까운 위치에서 코드를 실행하여 성능을 개선하고 지연 시간을 줄일 수 있습니다. 관리할 인프라는 없습니다.

AWS HealthImaging 뷰어 UI

프로그레시브 디코딩을 사용하여 AWS HealthImaging에 저장된 이미지 세트 메타데이터 속성과 이미지 프레임(픽셀 데이터)을 볼 수 있는 백엔드 인증을 갖춘 프런트엔드 UI를 배포하는 AWS Amplify 프로젝트입니다. 선택적으로 위의 타일 레벨 마커(TLM) 프록시 및/또는 Amazon CloudFront 전송 프로젝트를 통합하여 다른 방법을 사용하여 이미지 프레임을 로드할 수 있습니다.

AWS HealthImaging DICOMweb 프록시

HealthImaging 데이터 스토어에서 DICOMweb WADO-RS 및 QIDO-RS 엔드포인트를 활성화하여 웹 기반 의료 영상 뷰어 및 기타 DICOMweb 호환 앱을 지원하는 Python 기반 프로젝트입니다.

> **Note**
>
> 이 프로젝트는에 설명된 HealthImaging의 DICOMweb APIs 사용하지 않습니다AWS HealthImaging에서 DICOMweb 사용.

추가 샘플 프로젝트를 보려면 GitHub의 AWS HealthImaging 샘플을 참조하십시오.

AWS SDK에서이 서비스 사용

AWS 소프트웨어 개발 키트(SDKs)는 널리 사용되는 많은 프로그래밍 언어에 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예시 및 설명서를 제공합니다.

SDK 설명서	코드 예제
AWS SDK for C++	AWS SDK for C++ 코드 예제
AWS CLI	AWS CLI 코드 예제
AWS SDK for Go	AWS SDK for Go 코드 예제
AWS SDK for Java	AWS SDK for Java 코드 예제
AWS SDK for JavaScript	AWS SDK for JavaScript 코드 예제
AWS SDK for Kotlin	AWS SDK for Kotlin 코드 예제
AWS SDK for .NET	AWS SDK for .NET 코드 예제
AWS SDK for PHP	AWS SDK for PHP 코드 예제
AWS Tools for PowerShell	Tools for PowerShell 코드 예시
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 코드 예제
AWS SDK for Ruby	AWS SDK for Ruby 코드 예제
AWS SDK for Rust	AWS SDK for Rust 코드 예제
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP 코드 예제
AWS SDK for Swift	AWS SDK for Swift 코드 예제

예제 사용 가능 여부

필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예시를 요청하세요.

AWS HealthImaging 출시

다음 표는 AWS HealthImaging 서비스 및 설명서에 대한 기능 및 업데이트가 출시된 시기를 보여줍니다. 릴리스에 대한 자세한 내용은 링크된 주제를 참조하십시오.

변경 사항	설명	날짜
자동 데이터 구성, DICOMweb QIDO-RS 검색 및 DICOMweb WADO-RS 개선 사항	HealthImaging은 DICOM 표준의 환자, 연구 및 시리즈 수준 계층 구조에 따라 가져오기 시 DICOM P10 데이터를 자동으로 구성합니다. 자세한 내용은 가져오기 작업 이해를 참조하세요. HealthImaging은 DICOMweb QIDO-RS 표준에 따라 풍부한 검색을 지원합니다. 자세한 내용은 HealthImaging에서 DICOM 데이터 검색을 참조하세요. HealthImaging은 HealthImaging에서 DICOM 시리즈 메타데이터 가져오기에 설명된 대로 단일 API 작업을 통해 시리즈의 모든 DICOM 인스턴스에 대한 메타데이터 검색을 지원합니다.	2025년 5월 22일
이미지 세트 생성 시 더 적은 DICOM 요소 사용	HealthImaging은 수신 DICOM P10 객체를 이미지 세트로 그룹화할 때 사용되는 요소 수를 줄입니다. 자세한 내용은 이미지 세트란 무엇입니까?를 참조하세요.	2025년 1월 27일
StartDICOMImportJob에 대한 손실 지원	HealthImaging은 DICOM 손실 파일(1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50)	2024년 11월 1일

	및 이진 분할 파일을 원래 형식으로 가져오고 저장할 수 있도록 지원합니다. 자세한 내용은 지원되는 전송 구문을 참조하세요.	
DICOMweb 검색 APIs에 대한 손실 지원	HealthImaging은 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 및 1.2.840.10008.1.2.1(이진 분할만 해당)에 원본 형식 또는 명시적 VR Little Endian(1.2.840.10008.1.2.1)으로 저장된 이미지 및 인스턴스 검색을 지원합니다. 자세한 내용은 지원되는 전송 구문 및 DICOM 데이터 검색을 참조하세요.	2024년 11월 1일
디지털 리포지토리를 위한 더 빠른 가져오기	HealthImaging은 DICOM 디지털 리포지토리(WSI)에 대해 최대 6배 더 빠른 가져오기 작업을 지원합니다.	2024년 11월 1일
이진 분할 지원	HealthImaging은 DICOM 이진 분할 파일의 수집과 검색을 모두 지원합니다. 자세한 내용은 지원되는 전송 구문을 참조하세요.	2024년 11월 1일

이전 이미지 세트 버전 ID로 되돌리기	HealthImaging은 이전 이미지 세트 버전 ID로 되돌릴 `revertToVersionId` 파라미터를 제공합니다. 자세한 내용은 AWS HealthImaging API Reference에서 `revertToVersionId` 섹션을 참조하세요.	2024년 7월 24일

이미지 세트 수정을 위한 강제 기능	HealthImaging은 `Overrides` 데이터 형식에 선택적 `forced` 요청 파라미터를 제공합니다. 이 파라미터를 설정하면 환자, 연구 또는 시리즈 수준 메타데이터가 일치하지 않더라도 `UpdateImageSetMetadata` 및 `CopyImageSet` 작업이 강제로 적용됩니다. 자세한 내용은 AWS HealthImaging API 참조의 재정의를 참조하세요.
	
	• `UpdateImageSetMetadata` force functionality - HealthImaging은 다음 속성을 업데이트하기 위한 선택적 `force` 요청 파라미터를 도입합니다.
	  • `Tag.StudyInstanceUID`, `Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID`, 및 `Tag.StudyID`
	  • 인스턴스 수준 프라이빗 DICOM 데이터 요소 추가, 제거 또는 업데이트
	
	  자세한 내용은 AWS HealthImaging API 참조의 UpdateImageSetMetadata를 참조하세요. HealthImaging
	
	• `CopyImageSet` force functionality - HealthImaging은 이미지 세트를 복사하기 위한 선택적 `force` 요청 파
		2024년 7월 24일

	라미터를 도입합니다. 이 파라미터를 설정하면 환자, 연구 또는 시리즈 수준 메타데이터가 `sourceImageSet` 및 간에 일치하지 않더라도 `CopyImageSet` 작업이 강제로 수행됩니다`destinationImageSet` . 이러한 경우 일관되지 않은 메타데이터는에서 변경되지 않습니다`destinationImageSet` . 자세한 내용은 AWS HealthImaging API 참조의 CopyImageSet를 참조하세요. HealthImaging	
SOP 인스턴스의 하위 집합 복사	HealthImaging은 `CopyImageSet` 작업을 개선하므로에서 하나 이상의 SOP 인스턴스를 선택하여 `sourceImageSet` 에 복사할 수 있습니다`destinationImageSet` . 자세한 내용은 이미지 세트 복사를 참조하세요.	2024년 7월 24일
`GetDICOMInstanceMetadata` DICOM 인스턴스 메타데이터 반환	HealthImaging은 DICOM Part 10 메타데이터(`.json` 파일)를 반환하는 `GetDICOMInstanceMetadata` API를 제공합니다. 자세한 내용은 인스턴스 메타데이터 가져오기를 참조하세요.	2024년 7월 11일

GetDICOMInstanceFrames DICOM 인스턴스 프레임 반환용(픽셀 데이터)	HealthImaging은 DICOM Part 10 프레임(multipart 요청)을 반환하는 GetDICOMInstanceFrames API를 제공합니다. 자세한 내용은 인스턴스 프레임 가져오기를 참조하세요.	2024년 7월 11일

비표준 DICOM 데이터 가져오기에 대한 지원 향상	HealthImaging은 DICOM 표준과의 편차를 포함하는 데이터 가져오기를 지원합니다. 자세한 내용은 DICOM 요소 제약 조건을 참조하세요.
	• 다음 DICOM 데이터 요소는 최대 256자일 수 있습니다.
	  • `Patient's Name (0010,0010)`
	  • `Patient ID (0010,0020)`
	  • `Accession Number (0008,0050)`
	• `Study Instance UID`, , , , 및 에는 `Acquisition UID` 다음 구문 변형이 허용됩니다`Series Instance UID``Treatment Session UID`` Manufacturer’s Device Class UID`` Device UID`.
	  • UID의 첫 번째 요소는 0일 수 있습니다.
	  • UIDs 하나 이상의 선행 0으로 시작할 수 있습니다.
	  • UIDs 최대 256자까지 가능합니다.
		2024년 6월 28일

이벤트 알림	HealthImaging은 Amazon EventBridge와 통합되어 이벤트 기반 애플리케이션을 지원합니다. 자세한 내용은 EventBridge 사용을 참조하세요.	2024년 6월 5일

GetDICOMInstance DICOM 인스턴스 데이터 반환	HealthImaging은 DICOM Part 10 인스턴스 데이터(.dcm 파일)를 반환하는 GetDICOMInstance 서비스를 제공합니다. 자세한 내용은 인스턴스 가져오기를 참조하세요.	2024년 5월 15일
교차 계정 가져오기	HealthImaging은 지원되는 다른 리전에 있는 Amazon S3 버킷에서 데이터 가져오기를 지원합니다. 자세한 내용은 교차 계정 가져오기를 참조하세요.	2024년 5월 15일
이미지 세트 검색 개선 사항	HealthImaging SearchImageSets 작업은 다음과 같은 검색 개선 사항을 지원합니다. 자세한 내용은 이미지 세트 검색을 참조하세요. • UpdatedAt 및 검색에 대한 추가 지원 SeriesInstanceUID • 시작 시간과 종료 시간 간의 검색 • Ascending 또는를 기준으로 검색 결과 정렬 Descending • DICOM 시리즈 파라미터는 응답으로 반환됩니다.	2024년 4월 3일
가져오기의 최대 파일 크기 증가	HealthImaging은 가져오기 작업의 각 DICOM P10 파일에 대해 최대 4GB의 파일 크기를 지원합니다. 자세한 내용은 Service Quotas을 참조하세요.	2024년 3월 6일

JPEG Lossless 및 HTJ2K에 대한 전송 구문	HealthImaging은 작업 가져오기에 대해 다음과 같은 전송 구문을 지원합니다. 자세한 내용은 지원되는 전송 구문을 참조하세요. • 1.2.840.10008.1.2.4.57 — JPEG 무손실 비계층적(프로세스 14) • 1.2.840.10008.1.2.4.201 - 처리량이 많은 JPEG 2000 이미지 압축(무손실 전용) • 1.2.840.10008.1.2.4.202 - RPCL 옵션 이미지 압축을 사용하는 고처리량 JPEG 2000(무손실 전용) • 1.2.840.10008.1.2.4.203 — 처리량이 많은 JPEG 2000 이미지 압축	2024년 2월 16일
테스트된 코드 예제	HealthImaging 설명서는 Python, JavaScript, Java 및 C++용 AWS CLI 및 AWS SDKs 대해 테스트된 코드 예제를 제공합니다. 자세한 내용은 코드 예제를 참조하세요.	2023년 12월 19일
가져오기의 최대 파일 수 증가	HealthImaging은 단일 가져오기 작업에 대해 최대 5,000개의 파일을 지원합니다. 자세한 내용은 Service Quotas을 참조하세요.	2023년 12월 19일

가져오기를 위한 중첩 폴더	HealthImaging은 단일 가져오기 작업에 대해 최대 1만 개의 중첩 폴더를 지원합니다. 자세한 내용은 서비스 할당량을 참조하세요.	2023년 12월 1일
더 빠른 가져오기	HealthImaging은 지원되는 모든 리전에서 20배 더 빠른 가져오기를 제공합니다. 자세한 내용은 서비스 엔드포인트를 참조하세요.	2023년 12월 1일
AWS CloudFormation 지원	AWS HealthImaging은 데이터 스토어를 프로비저닝하기 위한 코드형 인프라(IaC)를 지원합니다. 자세한 내용은 를 사용하여 HealthImaging 리소스 생성을 AWS CloudFormation 참조하세요.	2023년 9월 21일
정식 출시	AWS HealthImaging은 미국 동부(버지니아 북부), 미국 서부(오레곤), 유럽(아일랜드) 및 아시아 태평양(시드니) 리전에서 정식으로 출시되었습니다. 자세한 내용은 서비스 엔드포인트를 참조하세요.	2023년 7월 26일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.