aws

Guia do desenvolvedor

AWS HealthImaging

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

AWS HealthImaging: Guia do desenvolvedor

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é a AWS HealthImaging? 1
Aviso importante 2
Recursos 2
Serviços relacionados 4
Acesso 4
HIPAA 5
Preços 6
Introdução 7
Conceitos 7
Datastore 7
Conjunto de imagens 8
Metadados 8
Quadro da imagem 8
Configurar 9
Inscreva-se para um Conta da AWS 9
Criar um usuário com acesso administrativo 10
Crie buckets do S3 11
Criar um datastore 12
Criar um usuário do IAM 12
Criar um perfil do IAM 13
Instale o AWS CLI 15
Tutorial 16
Gerenciar datastores 18
Criar um datastore 18
Obter propriedades do datastore 27
Listar datastore 35
Excluir um datastore 42
Usando DICOMweb 50
Armazenamento de instâncias com STOW-RS 50
Recuperação de dados com o WADO-RS 54
Recuperar uma instância 55
Recuperar metadados da instância 58
Recuperar metadados da série 59
Recuperar quadros 61

Recupere dados em massa ... 63
Pesquisando dados com o QIDO-RS ... 65
DICOMweb pesquisar APIs por HealthImaging ... 66
Tipos DICOMweb de consulta compatíveis para HealthImaging ... 67
Pesquise estudos ... 70
Pesquisar séries ... 72
Pesquisar instâncias ... 73
Autenticação OIDC ... 74
Como funciona a verificação de token ... 75
Requisitos e configuração ... 79
Importar dados do sistema de imagem ... 91
Noções básicas sobre as tarefas de importação ... 91
Como iniciar trabalho de trabalho de trabalho de trabalho ... 95
Como obter propriedades do trabalho de importação ... 104
Listar trabalhos de importação ... 111
Acessar conjuntos de imagens ... 118
Noções básicas sobre conjuntos de imagem ... 118
Pesquisar conjuntos de imagens ... 126
Obtendo propriedades do conjunto de imagens ... 152
Obtendo metadados do conjunto de imagens ... 158
Obtendo dados de pixels do conjunto de imagens ... 169
Modificar os conjuntos de imagens ... 178
Listando versões do conjunto de imagens ... 178
Atualização dos metadados do conjunto de imagens ... 186
Para atualizar os metadados de um conjunto de imagens primário ... 205
Para tornar uma imagem não primária, defina como primária ... 207
Copiar um conjunto de imagens ... 207
Excluir um conjunto de imagens ... 223
Marcar recursos ... 231
Marcar um recurso ... 231
Listar as tags de um recurso ... 237
Desmarcar um recurso ... 242
Exemplos de código ... 249
Conceitos básicos ... 250
Olá HealthImaging ... 251
Ações ... 256

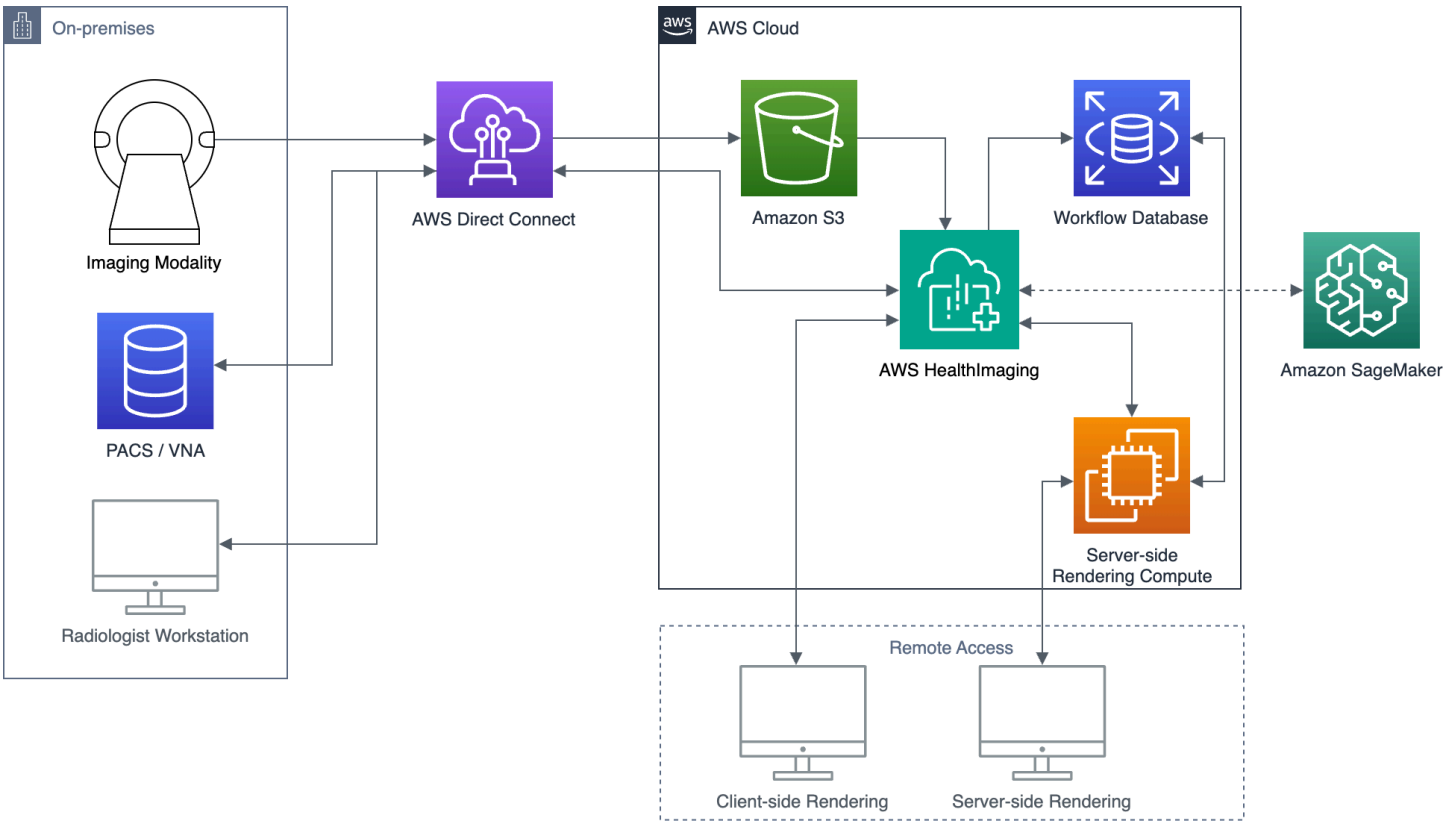
Cenários 411
Começar a usar conjuntos de imagens e quadros de imagem 412
Marcar um datastore 467
Marcar um conjunto de imagens 477
Monitoramento 488
CloudTrail (Chamadas de API) 488
Criar uma trilha 489
Noções básicas sobre entradas de log 491
CloudWatch (Métricas) 492
Métricas da API 493
HealthImaging Métricas 493
Dimensões 497
Acesso às métricas do 498
Configurando métricas 498
Visualizando HealthImaging métricas 498
Criar um alarme 499
EventBridge (Eventos) 499
HealthImaging eventos enviados para EventBridge 499
HealthImaging estrutura e exemplos de eventos 501
Segurança 518
Proteção de dados 519
Criptografia de dados 520
Privacidade do tráfego de rede 530
Gerenciamento de Identidade e Acesso 531
Público 531
Autenticação com identidades 532
Gerenciar o acesso usando políticas 533
Como a AWS HealthImaging trabalha com o IAM 535
Exemplos de políticas baseadas em identidade 541
AWS políticas gerenciadas 544
Prevenção do problema "confused deputy" entre serviços 547
Uso de perfis vinculados ao serviço 549
Solução de problemas 550
Validação de conformidade 552
Segurança da infraestrutura 553
Infraestrutura como código 553

HealthImaging e CloudFormation modelos 554
Saiba mais sobre CloudFormation 554
Endpoints da VPC 554
Considerações sobre endpoints da VPC do 555
Criar um endpoint da VPC 555
Criar uma política de endpoint da VPC 556
Importação entre contas 557
Resiliência 559
Referência 560
DICOM 560
Classes de SOP compatíveis 561
Normalização de metadados 561
Sintaxes de transferência compatíveis 566
Restrições de elementos de DICOM 569
Restrições de metadados de DICOM 570
HealthImaging 571
Endpoints e cotas 571
Limites de controle de utilização 577
Verificação de dados de pixel 579
Códigos de aviso 581
Bibliotecas de decodificação de quadros de imagem 585
Projetos de amostra 586
Trabalhando com AWS SDKs 588
Otimização de custo 590
Como funciona a hierarquização inteligente 590
Estimando o armazenamento de dados estruturados 591
Versões 593
..... dcxi

O que é a AWS HealthImaging?

HealthImaging A AWS é um serviço qualificado pela HIPAA que capacita prestadores de serviços de saúde, organizações de ciências biológicas e seus parceiros de software a armazenar, analisar e compartilhar imagens médicas na nuvem em escala de petabytes. HealthImaging Os casos de uso incluem:

- Imagens corporativas — armazene e transmita seus dados de imagens médicas diretamente da AWS nuvem, preservando o desempenho de baixa latência e a alta disponibilidade.
- Arquivamento de imagens de longo prazo — economize no arquivamento de imagens de longo prazo enquanto mantém o acesso à recuperação de imagens em menos de um segundo.
- Desenvolvimento de IA/ML — execute inferência de inteligência artificial e aprendizado de máquina (AI/ML) em seu arquivo de imagens com o suporte de outras ferramentas e serviços.
- Análise multimodal — Combine seus dados de imagens clínicas com AWS HealthLake (dados de saúde) e AWS HealthOmics (dados ômicos) para fornecer informações sobre medicina de precisão.



HealthImaging A AWS fornece acesso a dados de imagem (por exemplo, X-Ray, tomografia computadorizada, ressonância magnética, ultrassom) para que os aplicativos de imagens médicas criados na nuvem possam alcançar um desempenho que antes só era possível localmente. Com isso HealthImaging, você reduz os custos de infraestrutura executando seus aplicativos de imagens médicas em grande escala a partir de uma única cópia oficial de cada imagem médica recebida. Nuvem AWS

Tópicos

- Aviso importante
- Características da AWS HealthImaging
- AWS Serviços relacionados
- Acessando a AWS HealthImaging
- Elegibilidade e segurança de dados da HIPAA
- Preços

Aviso importante

HealthImaging A AWS não substitui o aconselhamento, diagnóstico ou tratamento médico profissional e não se destina a curar, tratar, mitigar, prevenir ou diagnosticar qualquer doença ou condição de saúde. Você é responsável por instituir a avaliação humana como parte de qualquer uso da AWS HealthImaging, inclusive em associação com qualquer produto de terceiros destinado a informar a tomada de decisões clínicas. A AWS só HealthImaging deve ser usada no atendimento ao paciente ou em cenários clínicos após análise por profissionais médicos treinados que apliquem um julgamento médico sólido.

Características da AWS HealthImaging

HealthImaging A AWS fornece os seguintes recursos.

Metadados DICOM fáceis de usar para desenvolvedores

A AWS HealthImaging simplifica o desenvolvimento de aplicativos ao retornar metadados DICOM em um formato amigável ao desenvolvedor. Depois de importar seus dados de imagem, os atributos individuais de metadados podem ser acessados usando palavras-chave fáceis de usar, em vez de números hexadecimais desconhecidos. group/element Os elementos DICOM em nível

de paciente, estudo e série são normalizados, eliminando a necessidade de os desenvolvedores de aplicações lidarem com inconsistências entre instâncias de SOP. Além disso, os valores dos atributos de metadados podem ser acessados diretamente nos tipos de runtime nativos.

Decodificação de imagem acelerada por SIMD

A AWS HealthImaging retorna quadros de imagem (dados em pixels) codificados como imagens JPEG 2000 (HTJ2K) de alto rendimento, um codec avançado de compressão de imagens. HTJ2O K aproveita os dados múltiplos de instrução única (SIMD) em processadores modernos para oferecer novos níveis de desempenho. HTJ2K é uma ordem de magnitude mais rápida que JPEG2 000 e pelo menos duas vezes mais rápida que todas as outras sintaxes de transferência DICOM. O WASM-SIMD pode ser utilizado para reduzir essa velocidade extrema a zero para visualizadores da web. Para obter mais informações, consulte Sintaxes de transferência compatíveis.

Verificação de dados de pixel

HealthImaging A AWS fornece verificação integrada de dados de pixels ao verificar o estado de codificação e decodificação sem perdas de cada imagem durante a importação. Para obter mais informações, consulte Verificação de dados de pixel.

Desempenho líder do setor

A AWS HealthImaging define um novo padrão para desempenho de carregamento de imagens graças à sua codificação eficiente de metadados, compactação sem perdas e acesso a dados com resolução progressiva. A codificação eficiente de metadados permite que visualizadores de imagens e algoritmos de IA entendam o conteúdo de um estudo DICOM sem precisar carregar os dados da imagem. As imagens são carregadas mais rapidamente sem comprometer a qualidade da imagem, graças à compressão avançada da imagem. A resolução progressiva permite um carregamento de imagens ainda mais rápido para miniaturas, regiões de interesse e dispositivos móveis de baixa resolução.

Importações DICOM escaláveis

HealthImaging As importações da AWS utilizam tecnologias modernas nativas da nuvem para importar vários estudos DICOM em paralelo. Os arquivos históricos podem ser importados rapidamente sem afetar as cargas de trabalho clínicas de novos dados. Para obter informações sobre instâncias de SOP compatíveis e sintaxes de transferência, consulte DICOM.

Hierarquia de dados DICOM gerenciada por serviços

A AWS organiza HealthImaging automaticamente os dados DICOM P10 na importação por elementos de dados DICOM de nível de paciente, estudo e série. O serviço organiza esses dados

DICOM em conjuntos de imagens que correspondem à série DICOM, simplificando os fluxos de trabalho pós-importação. A organização em nível de Estudo e Série é mantida à medida que novos dados são importados.

DICOMweb Compatibilidade com APIs

HealthImaging A AWS oferece DICOMweb conformidade APIs para simplificar as integrações e permitir a interoperabilidade com aplicativos existentes. O serviço também oferece soluções nativas em nuvem APIs que permitem ações não suportadas pelo DICOMweb padrão, como operações de atualização de metadados.

AWS Serviços relacionados

A AWS HealthImaging oferece uma forte integração com outros AWS serviços. O conhecimento dos serviços a seguir é útil para aproveitar totalmente HealthImaging.

- AWS Identity and Access Management— Use o IAM para gerenciar com segurança as identidades e o acesso aos recursos. HealthImaging
- Amazon Simple Storage Service — Use o Amazon S3 como uma área de armazenamento para importar dados DICOM. HealthImaging
- Amazon CloudWatch — Use CloudWatch para observar e monitorar HealthImaging recursos.
- AWS CloudTrail— Use CloudTrail para rastrear a atividade HealthImaging do usuário e o uso da API.
- AWS CloudFormation— Use CloudFormation para implementar modelos de infraestrutura como código (IaC) para criar recursos em HealthImaging.
- AWS PrivateLink— Use a Amazon VPC para estabelecer conectividade entre HealthImaging a Amazon Virtual Private Cloud sem expor dados à Internet.
- Amazon EventBridge — Use EventBridge para criar aplicativos escaláveis e orientados por eventos criando regras que direcionam HealthImaging eventos para destinos.

Acessando a AWS HealthImaging

Você pode acessar a AWS HealthImaging usando o Console de gerenciamento da AWS, AWS Command Line Interface AWS SDKs e. Este guia fornece instruções processuais para o Console de gerenciamento da AWS e exemplos de código para e. AWS CLI AWS SDKs

Console de gerenciamento da AWS

O Console de gerenciamento da AWS fornece uma interface de usuário baseada na web para gerenciamento HealthImaging e seus recursos associados. Se você se inscreveu em uma AWS conta, pode entrar no HealthImaging console.

AWS Command Line Interface (AWS CLI)

O AWS CLI fornece comandos para um amplo conjunto de AWS produtos e é compatível com Windows, Mac e Linux. Para obter mais informações, consulte o Guia do usuário do AWS Command Line Interface .

AWS SDKs

AWS SDKs forneça bibliotecas, exemplos de código e outros recursos para desenvolvedores de software. Essas bibliotecas fornecem funções básicas que automatizam tarefas como a assinatura criptografada das solicitações, novas tentativas de solicitações e tratamento das respostas de erro. Para obter mais informações, consulte Ferramentas para desenvolver AWS.

Solicitações HTTP

Você pode chamar HealthImaging ações usando solicitações HTTP, mas deve especificar pontos de extremidade diferentes, dependendo do tipo de ação que está sendo usada. Para obter mais informações, consulte Ações de API compatíveis para solicitações HTTP.

Elegibilidade e segurança de dados da HIPAA

Este é um serviço qualificado da HIPAA. Para obter mais informações sobre AWS a Lei de Portabilidade e Responsabilidade de Seguros de Saúde dos EUA de 1996 (HIPAA) e o uso de AWS serviços para processar, armazenar e transmitir informações de saúde protegidas (PHI), consulte Visão geral da HIPAA.

As conexões que HealthImaging contêm PHI e informações de identificação pessoal (PII) devem ser criptografadas. Por padrão, todas as conexões devem HealthImaging usar HTTPS sobre TLS. HealthImaging armazena conteúdo criptografado do cliente e opera de acordo com o Modelo de Responsabilidade AWS Compartilhada.

Para obter informações sobre conformidade, consulte Validação de conformidade para a AWS HealthImaging.

Preços

HealthImaging ajuda você a automatizar o gerenciamento do ciclo de vida dos dados clínicos com hierarquização inteligente. Para obter mais informações, consulte Otimização de custo.

Para obter informações gerais sobre preços, consulte os HealthImaging preços da AWS. Para estimar os custos, use a calculadora HealthImaging de preços da AWS.

Comece a usar a AWS HealthImaging

Para começar a usar a AWS HealthImaging, configure uma AWS conta e crie um AWS Identity and Access Management usuário. Para usar o AWS CLIou o AWS SDKs, você deve instalá-los e configurá-los.

Depois de aprender sobre HealthImaging conceitos e configuração, um breve tutorial com exemplos de código está disponível para ajudar você a começar.

Tópicos

- HealthImaging Conceitos da AWS
- Configurando a AWS HealthImaging
- HealthImaging Tutorial da AWS

HealthImaging Conceitos da AWS

A terminologia e os conceitos a seguir são fundamentais para sua compreensão e uso da AWS HealthImaging.

Conceitos

- Datastore
- Conjunto de imagens
- Metadados
- Quadro da imagem

Datastore

Um datastore é um repositório de dados de imagens médicas que reside em um único Região da AWS. Uma AWS conta pode ter zero ou muitos armazenamentos de dados. Um datastore tem sua própria chave de criptografia do AWS KMS , portanto, os dados em um datastore podem ser isolados física e logicamente dos dados em outros datastores. Os datastores oferecem suporte ao controle de acesso usando perfis, permissões e controle de acesso baseado em atributos do IAM.

Para obter mais informações, consulte Gerenciar datastores e Otimização de custo.

Conjunto de imagens

Um conjunto de imagens é um AWS conceito que define um mecanismo de agrupamento abstrato para otimizar dados de imagens médicas relacionados. Quando você importa seus dados de imagem DICOM P10 para um armazenamento de HealthImaging dados da AWS, eles são transformados em conjuntos de imagens compostos por metadados e quadros de imagem (dados em pixels). HealthImaging tenta organizar os dados importados de acordo com a hierarquia DICOM de Estudo, Série e Instância. As instâncias DICOM que são adicionadas com sucesso à hierarquia HealthImaging gerenciada são indicadas como conjuntos de imagens primárias. A importação de dados DICOM P10 pode: criar um novo conjunto de imagens primárias; mesclar instâncias em um conjunto de imagens primárias existente se as instâncias já existirem na coleção primária; ou, no caso de conflitos de elementos de metadados, criar um novo conjunto de imagens não primárias.

Para obter mais informações, consulte Importar dados do sistema de imagem e Noções básicas sobre conjuntos de imagem.

Metadados

Metadados são os atributos que não são pixels que existem em um conjunto de imagens. Para o DICOM, isso inclui dados demográficos do paciente, detalhes do procedimento e outros parâmetros específicos da aquisição. A AWS HealthImaging separa o conjunto de imagens em metadados e quadros de imagem (dados em pixels) para que os aplicativos possam acessá-lo rapidamente. Isso é útil para visualizadores de imagens, análises e casos de AI/ML uso que não exigem dados de pixels. Os dados DICOM se normalizam nos níveis do paciente, do estudo e da série, eliminando inconsistências. Isso simplifica o uso dos dados, aumenta a segurança e melhora o desempenho do acesso.

Para obter mais informações, consulte Obtendo metadados do conjunto de imagens e Normalização de metadados.

Quadro da imagem

Um quadro de imagem são os dados de pixel que existem em um conjunto de imagens para formar uma imagem médica 2D. Alguns arquivos mantêm a codificação da sintaxe de transferência original durante a importação, enquanto outros são transcodificados. Os armazenamentos de dados da Amazon Web Services podem ser configurados para transcodificar quadros de imagem sem perdas para JPEG 2000 (HTJ2K) de alto rendimento sem perdas ou JPEG 2000 sem perdas. Se um quadro de imagem estiver codificado em HTJ2 K ou JPEG 2000, ele deverá ser decodificado antes de

ser visualizado em um visualizador de imagens. Para obter mais informações, consulte Sintaxes de transferência compatíveis, Obtendo dados de pixels do conjunto de imagens e Bibliotecas de decodificação de quadros de imagem.

Configurando a AWS HealthImaging

Você deve configurar seu AWS ambiente antes de usar a AWS HealthImaging. Os tópicos a seguir são pré-requisitos para o tutorial localizado na próxima seção.

Tópicos

- Inscreva-se para um Conta da AWS
- Criar um usuário com acesso administrativo
- Crie buckets do S3
- Criar um datastore
- Crie um usuário do IAM com permissão de acesso HealthImaging total
- Criar um perfil do IAM para importação
- Instale o AWS CLI (opcional)

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica ou uma mensagem de texto e inserir um código de verificação pelo teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS Centro de Identidade do AWS IAM, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login Console de gerenciamento da AWScomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, insira a senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Fazer login como usuário-raiz no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilita o Centro de Identidade do IAM.

   Para obter instruções, consulte Habilitar o Centro de Identidade do AWS IAM no Guia do usuário do Centro de Identidade do AWS IAM .
2. No Centro de Identidade do IAM, conceda o acesso administrativo a um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia Centro de Identidade do AWS IAM do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Criar um conjunto de permissões no Guia do usuário do Centro de Identidade do AWS IAM .
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de logon único ao grupo.

   Para obter instruções, consulte Adicionar grupos no Guia do usuário do Centro de Identidade do AWS IAM .

Crie buckets do S3

Para importar dados DICOM P10 para a AWS HealthImaging, dois buckets Amazon S3 são recomendados. O bucket de entrada do Amazon S3 armazena os dados DICOM P10 a serem importados e HealthImaging lê desse bucket. O bucket de saída do Amazon S3 armazena os resultados de processamento do trabalho de importação e HealthImaging grava nesse bucket. Para uma representação visual disso, veja o diagrama em Noções básicas sobre as tarefas de importação.

> **Note**
>
> Devido à política AWS Identity and Access Management (IAM), seus nomes de bucket do Amazon S3 devem ser exclusivos. Para obter mais informações, consulte as Regras para nomear buckets no Guia do usuário do Amazon Simple Storage Service.

Para fins deste guia, especificamos os seguintes buckets de entrada e saída do Amazon S3 no perfil do IAM para importação.

- Bucket de entrada: `arn:aws:s3:::amzn-s3-demo-source-bucket`
- Bucket de saída: `arn:aws:s3:::amzn-s3-demo-logging-bucket`

Para obter informações adicionais, consulte Criar um bucket no Guia do usuário do Amazon S3.

Criar um datastore

Quando você importa seus dados de imagens médicas, o armazenamento de HealthImaging dados da AWS armazena os resultados dos seus arquivos DICOM P10 transformados, chamados de conjuntos de imagens. Para uma representação visual disso, veja o diagrama em Noções básicas sobre as tarefas de importação.

> **Tip**
>
> Um `datastoreID` é gerado quando você cria um datastore. Você deve usar a `datastoreID` ao concluir o trust relationship para importação posteriormente nesta seção.

Para criar um datastore, veja Criar um datastore.

Crie um usuário do IAM com permissão de acesso HealthImaging total

> **Prática recomendada**
>
> Sugerimos que você crie usuários do IAM separados para diferentes necessidades, como importação, acesso a dados e gerenciamento de dados. Isso se alinha à Conceder acesso com privilégio mínimo no Well-Architected Framework da AWS .
> Para os fins do Tutorial, na próxima seção, você usará um único usuário do IAM.

Para criar um usuário do IAM

1. Siga as instruções para criar um usuário do IAM em sua AWS conta no Guia do usuário do IAM. Considere nomear o usuário `ahiadmin` (ou similar) para fins de esclarecimento.
2. Atribua a política de IAM gerenciada do `AWSHealthImagingFullAccess` ao usuário do IAM. Para obter mais informações, consulte AWS política gerenciada: AWSHealth ImagingFullAccess.

   > **Note**
   >
   > As permissões do IAM podem ser reduzidas. Para obter mais informações, consulte AWS políticas gerenciadas para a AWS HealthImaging.

Criar um perfil do IAM para importação

> **Note**
>
> As instruções a seguir se referem a uma função AWS Identity and Access Management (IAM) que concede acesso de leitura e gravação aos buckets do Amazon S3 para importar seus dados DICOM. Embora o perfil seja necessário para o tutorial da próxima seção, recomendamos que você adicione permissões do IAM aos usuários, grupos e funções usando AWS políticas gerenciadas para a AWS HealthImaging, porque elas são mais fáceis de usar do que escrever todas as políticas manualmente.

Um perfil do IAM é uma identidade do IAM que você pode criar em sua conta que tem permissões específicas. Para iniciar um trabalho de importação, o perfil do IAM que chama a ação `StartDICOMImportJob` deve ser anexado a uma política de usuário que conceda acesso aos buckets do Amazon S3 usados para ler seus dados DICOM P10 e armazenar os resultados do processamento do trabalho de importação. Também deve ser atribuída uma relação de confiança (política) que permita HealthImaging à AWS assumir a função.

Para criar um perfil do IAM para importação

1. Com o Console do IAM, crie um perfil e dê o nome de `ImportJobDataAccessRole`. Você usa esse perfil para o tutorial na próxima seção. Para obter mais informações, consulte Criar perfis do IAM no Guia do usuário do IAM.

   > **Tip**
   >
   > Para fins deste guia, os exemplos de código em Como iniciar trabalho de trabalho de trabalho de trabalho fazem referência ao perfil do IAM `ImportJobDataAccessRole`.

2. Anexe uma política de permissões do IAM ao perfil do IAM. Essa política de permissões concede acesso aos buckets de entrada e saída do Amazon S3. Anexe a política de permissões a seguir ao perfil `ImportJobDataAccessRole` do IAM.

   JSON

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

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

3. Anexe a seguinte relação de confiança (política) ao perfil `ImportJobDataAccessRole` do IAM. A política de confiança exige a `datastoreId` gerada quando você concluiu a seção Criar um datastore. O tutorial a seguir a este tópico pressupõe que você esteja usando um armazenamento de HealthImaging dados da AWS, mas com buckets Amazon S3 específicos para armazenamento de dados, funções do IAM e políticas de confiança.

> **Note**
>
> O `Condition` bloqueio nessa política de confiança ajuda a evitar o confuso problema do deputado, garantindo que somente seu armazenamento de HealthImaging dados

específico da AWS possa ser acessado. Para obter mais informações sobre essa medida de segurança, consulte Prevenção de delegados confusos entre serviços em HealthImaging.

JSON

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

Para saber mais sobre como criar e usar políticas do IAM com a AWS HealthImaging, consulteIdentity and Access Management para AWS HealthImaging.

Para saber mais sobre perfis do IAM, consulte Perfis do IAM no Manual do usuário do IAM. Para saber mais sobre as políticas do IAM e permissões em geral, consulte Permissões e políticas do IAM no Guia do usuário do IAM.

Instale o AWS CLI (opcional)

O procedimento a seguir é necessário se você estiver usando o AWS Command Line Interface. Se você estiver usando o Console de gerenciamento da AWS ou AWS SDKs, você pode pular o procedimento a seguir.

Para configurar o AWS CLI

1. Faça download e configure a AWS CLI. Para obter instruções, consulte os seguintes tópicos no Guia do usuário do AWS Command Line Interface .

   - Instalando ou atualizando a versão mais recente do AWS CLI

- Começando com o AWS CLI

2. No AWS CLI `config` arquivo, adicione um perfil nomeado para o administrador. Você usa esse perfil ao executar os AWS CLI comandos. De acordo com o princípio de segurança do privilégio mínimo, recomendamos que você crie um perfil do IAM separado com privilégios específicos para as tarefas que estão sendo executadas. Para obter mais informações sobre perfis nomeados, consulte Configurações de arquivos de configuração e credenciais no Guia do usuário da AWS Command Line Interface .

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

3. Verifique a configuração usando o comando `help`.

```
aws medical-imaging help
```

Se o AWS CLI estiver configurado corretamente, você verá uma breve descrição da AWS HealthImaging e uma lista dos comandos disponíveis.

HealthImaging Tutorial da AWS

Objetivo

O objetivo deste tutorial é importar binários DICOM P10 (`.dcm`arquivos) para um armazenamento de HealthImaging dados da AWS e transformá-los em conjuntos de imagens compostos por metadados e quadros de imagem (dados de pixels). Depois de importar os dados DICOM, você usa ações nativas da HealthImaging nuvem para acessar os conjuntos de imagens, os metadados e os quadros de imagem com base na sua preferência de acesso.

Pré-requisitos

Todos os procedimentos listados em Configurar são necessários para concluir este tutorial.

Etapas do tutorial

1. Iniciar trabalho de importação
2. Obter as propriedades do trabalho de importação

3. Pesquisar conjuntos de imagens
4. Obter propriedades do conjunto de imagens
5. Obter metadados do conjunto de imagens
6. Obter dados de pixels do conjunto de imagens
7. Excluir datastore

Gerenciamento de datastores com a AWS HealthImaging

Com a AWS HealthImaging, você cria e gerencia armazenamentos de dados para recursos de imagens médicas. Os tópicos a seguir descrevem como usar ações nativas da HealthImaging nuvem para criar, descrever, listar e excluir armazenamentos de dados usando o Console de gerenciamento da AWS AWS CLI, AWS SDKs e.

> **Note**
>
> O último tópico deste capítulo é sobre otimização de custos. Depois de importar seus dados de imagens médicas para um armazenamento de HealthImaging dados, eles se movem automaticamente entre dois níveis de armazenamento com base no tempo e no uso. Os níveis de armazenamento têm níveis de preços diferentes, por isso é importante entender o processo de movimentação de níveis e os HealthImaging recursos que são reconhecidos para fins de cobrança.

Tópicos

- Criar um datastore
- Obter propriedades do datastore
- Listar datastore
- Excluir um datastore

Criar um datastore

Use a `CreateDatastore` ação para criar um armazenamento de HealthImaging dados da AWS para importar arquivos DICOM P10. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `CreateDatastore`a AWS HealthImaging API Reference. Ao criar um armazenamento de dados, você pode selecionar a sintaxe de transferência padrão que a AWS HealthImaging usou para transcodificar e armazenar quadros de imagem sem perdas. Essa configuração não pode ser alterada após a criação do armazenamento de dados.

JPEG 2000 (HTJ2K) de alto rendimento

HTJ2K (High Throughput JPEG 2000) é o formato de armazenamento padrão para HealthImaging datastores. É uma extensão do padrão JPEG 2000 que oferece desempenho de codificação e decodificação significativamente aprimorado. Quando você cria um armazenamento de dados sem especificar um`–lossless-storage-format`, HealthImaging usa automaticamente K. HTJ2 Consulte a AWS CLI e a SDKs seção abaixo para criar um armazenamento de dados usando K. HTJ2

JPEG 2000 sem perdas

A codificação JPEG 2000 Lossless permite a criação de armazenamentos de dados que persistem e recuperam quadros de imagem sem perdas no formato JPEG 2000 sem transcodificação, permitindo uma recuperação de menor latência para aplicativos que exigem JPEG 2000 Lossless (DICOM Transfer Syntax UID 1.2.840.10008.1.2.4.90). Consulte para obter mais detalhes. Sintaxes de transferência compatíveis Consulte a AWS CLI e a SDKs seção abaixo para criar um armazenamento de dados usando o formato JPEG 2000 sem perdas.

> **Importante**
>
> - Não nomeie datastores com informações de saúde protegidas (PHI), informações de identificação pessoal (PII) nem outras informações confidenciais ou sigilosas.
> - O console da AWS oferece suporte à criação de armazenamentos de dados com configurações padrão. Use a AWS CLI ou o AWS SDK para criar um armazenamento de dados com uma especificação opcional. `–lossless-storage-format`

Criar um datastore

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Criar armazenamento de dados do HealthImaging console.
2. Em Detalhes, em Nome do datastore, insira um nome para seu datastore.
3. Em Criptografia de dados, escolha uma AWS KMS chave para criptografar seus recursos. Para obter mais informações, consulte Proteção de dados na AWS HealthImaging.
4. Em Tags: opcional, você pode adicionar tags ao seu datastore ao criá-lo. Para obter mais informações, consulte Marcar um recurso.
5. Selecione Criar datastore.

AWS CLI and SDKs

Bash

AWS CLI com script Bash

```
##############################################################################
# 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
}
```

- Para obter detalhes da API, consulte CreateDatastorena Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: criar um datastore

O exemplo de código `create-datastore` a seguir cria um armazenamento de dados com o nome de `my-datastore`. Quando você cria um armazenamento de dados sem especificar um`--lossless-storage-format`, o AWS HealthImaging padrão é HTJ2 K (High Throughput JPEG 2000).

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

Saída:

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

Exemplo 2: criar um datastore com o formato de armazenamento JPEG 2000 Lossless

Um datastore configurado com o formato de armazenamento JPEG 2000 Lossless transcodificará e persistirá quadros de imagem sem perdas no formato JPEG 2000. Os quadros de imagem podem então ser recuperados no JPEG 2000 Lossless sem transcodificação. O exemplo de código `create-datastore` a seguir cria um datastore configurado para o formato de armazenamento JPEG 2000 Lossless com o nome `my-datastore`.

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

```
    --datastore-name "my-datastore" \
    --lossless-storage-format JPEG_2000_LOSSLESS
```

Saída:

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

Para obter mais informações, consulte Criação de um armazenamento de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte CreateDatastorena Referência de AWS CLI Comandos.

Java

SDK para 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 "";
   }
```

- Para obter detalhes da API, consulte CreateDatastorea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte CreateDatastorea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a CreateDatastoreReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a CreateDatastorereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Obter propriedades do datastore

Use a `GetDatastore` ação para recuperar as propriedades do armazenamento de HealthImaging dados da AWS. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `GetDatastore`a AWS HealthImaging API Reference.

Para obter propriedades do datastore

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta. Na seção Detalhes, todas as propriedades do datastore estão disponíveis. Para visualizar conjuntos de imagens, importações e tags associados, escolha a guia aplicável.

AWS CLI and SDKs

Bash

AWS CLI com script Bash

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

```
# 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
}
```

- Para obter detalhes da API, consulte GetDatastorena Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: obter as propriedades de um datastore

O exemplo de código `get-datastore` a seguir obtém as propriedades de um armazenamento de dados.

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

Saída:

```
{
    "datastoreProperties": {
        "datastoreId": "12345678901234567890123456789012",
        "datastoreName": "TestDatastore123",
        "datastoreStatus": "ACTIVE",
        "losslessStorageFormat": "HTJ2K"
        "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
        "createdAt": "2022-11-15T23:33:09.643000+00:00",
        "updatedAt": "2022-11-15T23:33:09.643000+00:00"
    }
}
```

Exemplo 2: Para configurar as propriedades do armazenamento de dados para JPEG2 000

O exemplo de código `get-datastore` a seguir obtém as propriedades de um datastore configurado para o formato de armazenamento JPEG 2000 Lossless.

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

Saída:

```
{
    "datastoreProperties": {
        "datastoreId": "12345678901234567890123456789012",
        "datastoreName": "TestDatastore123",
        "datastoreStatus": "ACTIVE",
        "losslessStorageFormat": "JPEG_2000_LOSSLESS",
        "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
        "createdAt": "2022-11-15T23:33:09.643000+00:00",
        "updatedAt": "2022-11-15T23:33:09.643000+00:00"
    }
}
```

Para obter mais informações, consulte Obter propriedades do armazenamento de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetDatastorena Referência de AWS CLI Comandos.

Java

SDK para 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;
    }
```

- Para obter detalhes da API, consulte GetDatastorea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

```
  // }
  return response.datastoreProperties;
};
```

- Para obter detalhes da API, consulte GetDatastorea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a GetDatastoreReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        oo_result = lo_mig->getdatastore( iv_datastoreid = iv_datastore_id ).
        DATA(lo_properties) = oo_result->get_datastoreproperties( ).
        DATA(lv_name) = lo_properties->get_datastorename( ).
        DATA(lv_status) = lo_properties->get_datastorestatus( ).
        MESSAGE 'Data store properties retrieved.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Data store not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
```

```
    ENDTRY.
```

- Para obter detalhes da API, consulte a GetDatastorereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Listar datastore

Use a `ListDatastores` ação para listar os armazenamentos de dados disponíveis na AWS HealthImaging. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `ListDatastores`a AWS HealthImaging API Reference.

Para listar datastores

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

- Abra a página Armazenamentos de dados do HealthImaging console.

  Todos os datastores estão listados na seção Datastore.

AWS CLI and SDKs

Bash

AWS CLI com script Bash

```
##############################################################################
# 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
}
```

- Para obter detalhes da API, consulte ListDatastoresem Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para listar armazenamentos de dados

O exemplo de código `list-datastores` a seguir lista os armazenamentos de dados disponíveis.

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

Saída:

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

Para obter mais informações, consulte Listar armazenamentos de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte ListDatastoresem Referência de AWS CLI Comandos.

Java

SDK para 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;
    }
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

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

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

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

  return datastoreSummaries;
};
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a ListDatastoresReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a ListDatastoresreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Excluir um datastore

Use a `DeleteDatastore` ação para excluir um armazenamento de HealthImaging dados da AWS. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da

AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `DeleteDatastore`a AWS HealthImaging API Reference.

> **Note**
>
> Antes que um datastore possa ser excluído, você deve primeiro excluir todos os conjuntos de imagens nele contidos. Para obter mais informações, consulte Excluir um conjunto de imagens.

Para excluir um datastore

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.
3. Escolha Excluir.

   A página Excluir datastore é aberta.
4. Para confirmar a exclusão do datastore, insira o nome do datastore no campo de entrada de texto.
5. Escolha Excluir datastore.

AWS CLI and SDKs

Bash

AWS CLI com script Bash

```
##############################################################################
# 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
}
```

- Para obter detalhes da API, consulte DeleteDatastoreem Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para excluir um armazenamento de dados

O exemplo de código `delete-datastore` a seguir exclui um armazenamento de dados.

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

Saída:

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

Para obter mais informações, consulte Excluindo um armazenamento de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte DeleteDatastoreem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte DeleteDatastorea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte DeleteDatastorea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a DeleteDatastoreReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->deletedatastore( iv_datastoreid = iv_datastore_id ).
    MESSAGE 'Data store deleted.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict. Data store may contain resources.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Data store not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a DeleteDatastorereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Usando DICOMweb com a AWS HealthImaging

Você pode recuperar objetos DICOM da AWS HealthImaging usando uma representação de DICOMweb APIs, que é baseada na web APIs e segue o padrão DICOM para imagens médicas. Essa funcionalidade permite que você interopere com sistemas que utilizam binários DICOM Parte 10 e, ao mesmo tempo, aproveite as ações nativas HealthImaging da nuvem. O foco deste capítulo é como usar a implementação HealthImaging de DICOMweb APIs para retornar DICOMweb respostas.

> **Importante**
>
> HealthImaging armazena dados DICOM como conjuntos de imagens. Use ações HealthImaging nativas da nuvem para gerenciar e recuperar conjuntos de imagens. HealthImaging's DICOMweb APIs podem ser usados para retornar informações do conjunto de imagens com respostas em DICOMweb conformidade com -.
> Os APIs itens listados neste capítulo foram criados em conformidade com o DICOMwebpadrão para imagens médicas baseadas na web. Por serem representações de DICOMweb APIs, não são oferecidas por meio de AWS CLI AWS SDKs e.

Tópico

- Armazenamento de instâncias com STOW-RS
- Recuperando dados DICOM de HealthImaging
- Pesquisando dados DICOM em HealthImaging
- Autenticação OIDC para DICOMweb APIs

Armazenamento de instâncias com STOW-RS

HealthImaging A AWS oferece uma representação do `DICOMweb STOW-RS` APIs para importar dados. Use-os APIs para armazenar de forma síncrona dados DICOM em seu armazenamento de HealthImaging dados.

A tabela a seguir descreve as HealthImaging representações do DICOMweb STOW-RS APIs disponíveis para importação de dados.

HealthImaging representações do DICOMweb STOW-RS APIs

Name (Nome)	Description
`StoreDICOM`	Armazene uma ou mais instâncias em um armazenamento HealthImaging de dados.
`StoreDICOMStudy`	Armazene uma ou mais instâncias correspondentes a um UID de instância de estudo especificado em um armazenamento HealthImaging de dados.

Os dados importados com as `StoreDICOMStudy` ações `StoreDICOM` e serão organizados como novos conjuntos de imagens primárias ou adicionados aos conjuntos de imagens primárias existentes, usando a mesma lógica dos trabalhos de importação assíncrona. Se os elementos de metadados dos dados DICOM P10 recém-importados entrarem em conflito com os conjuntos de imagens primárias existentes, os novos dados serão adicionados aos conjuntos de imagens não primárias.

> **Note**
>
> - Essas ações suportam o upload de até 1 GB de dados DICOM por solicitação.
> - A resposta da API estará no formato JSON, em conformidade com o padrão STOW-RS. DICOMweb

Para iniciar uma solicitação StoredIcom

1. Colete sua região da AWS e HealthImaging `datastoreId` o nome do arquivo DICOM P10.
2. Crie uma URL para a solicitação do formulário: `https://dicom-medical-imaging.`*`region`*`.amazonaws.com/datastore/`*`datastore-id`*`/studies`
3. Determine o tamanho do conteúdo do arquivo DICOM P10 usando seu comando preferido, por exemplo. `$(stat -f %z $FILENAME)`
4. Prepare e envie sua solicitação. `StoreDICOM`usa uma solicitação HTTP POST com o protocolo de assinatura AWS Signature versão 4.

Example Exemplo 1: Para armazenar um arquivo DICOM P10 usando a ação **StoreDICOM**

Shell

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

Example Exemplo 2: Para armazenar um arquivo DICOM P10 usando a ação **StoreDICOMStudy**

A única diferença entre StoredIcom e Store DICOMStudy é que um UID de instância de estudo é passado como um parâmetro para o StoreDICOMStudy, e as instâncias carregadas devem ser membros do estudo especificado.

Shell

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

Example Exemplo 3: Para armazenar arquivos DICOM P10 com uma carga HTTP de várias partes

Vários arquivos P10 podem ser carregados com uma única ação de upload de várias partes. Os comandos de shell a seguir demonstram como montar uma carga de várias partes contendo dois arquivos P10 e carregá-la com a ação. `StoreDICOM`

Shell

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

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

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

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

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

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

# Delete the payload file
```

```
rm $FILENAME
```

Recuperando dados DICOM de HealthImaging

HealthImaging A AWS oferece representações de DICOMweb WADO-RS APIs para recuperar dados nos níveis de série e instância. Com eles APIs, é possível recuperar todos os metadados de uma série DICOM de um armazenamento de dados. HealthImaging Também é possível recuperar uma instância DICOM, metadados da instância DICOM e quadros de instância DICOM (dados em pixels). HealthImagingOs DICOMweb WADO-RS APIs oferecem flexibilidade na forma como você recupera dados armazenados HealthImaging e fornece interoperabilidade com aplicativos legados.

> **Importante**
>
> HealthImaging armazena dados DICOM como conjuntos de imagens. Use ações nativas da HealthImaging nuvem para gerenciar e recuperar conjuntos de imagens. HealthImaging's DICOMweb APIs podem ser usados para retornar informações do conjunto de imagens com respostas em DICOMweb conformidade com -.
> Os APIs itens listados nesta seção foram criados em conformidade com o padrão DICOMweb (WADO-RS) para imagens médicas baseadas na web. Por serem representações de DICOMweb APIs, não são oferecidas por meio de AWS CLI AWS SDKs e.

A tabela a seguir descreve todas as HealthImaging representações do DICOMweb WADO-RS APIs disponíveis para recuperação de dados. HealthImaging

HealthImaging representações da DICOMweb WADO-RS APIs

Name (Nome)	Description
GetDICOMSeriesMetadata	Recupere metadados de instância DICOM (.jsonarquivo) para uma série DICOM em um armazenamento de HealthImaging dados especificando o estudo e a série UIDs associados a um recurso. Consulte Recuperar metadados da série.

Name (Nome)	Description
`GetDICOMInstance`	Recupere uma instância DICOM (`.dcm`arquivo) de um armazenamento de HealthImaging dados especificando a série, o estudo e a instância UIDs associados a um recurso. Consulte Recuperar uma instância.
`GetDICOMInstanceMetadata`	Recupere metadados de instância DICOM (`.json`arquivo) de uma instância DICOM em um armazenamento de HealthImaging dados especificando a série, o estudo e a instância UIDs associados a um recurso. Consulte Recuperar metadados da instância.
`GetDICOMInstanceFrames`	Recupere quadros de imagem individuais ou em lote (`multipart` solicitação) de uma instância DICOM em um armazenamento de HealthImaging dados especificando o UID da série, o UID do estudo, a instância e os números dos quadros UIDs associados a um recurso. Consulte Recuperar quadros.

Tópicos

- Obtendo uma instância DICOM de HealthImaging
- Obtendo metadados da instância DICOM de HealthImaging
- Obtendo metadados da série DICOM de HealthImaging
- Obtendo quadros de instância DICOM de HealthImaging
- Obtendo dados em massa DICOM de HealthImaging

Obtendo uma instância DICOM de HealthImaging

Use a `GetDICOMInstance` ação para recuperar uma instância DICOM (`.dcm`arquivo) de um armazenamento de HealthImaging dados especificando a série, o estudo e a instância UIDs associados ao recurso. A API retornará somente instâncias dos conjuntos de imagens primários,

a menos que o parâmetro opcional do conjunto de imagens seja fornecido. Você pode recuperar qualquer instância (de conjuntos de imagens primárias ou não primárias) no armazenamento de dados especificando o `imageSetId` como um parâmetro de consulta. Os dados DICOM podem ser recuperados em sua sintaxe de transferência armazenada ou no formato não compactado (ELE).

Para obter uma instância DICOM () **`.dcm`**

1. Colete HealthImaging `datastoreId` e defina `imageSetId` parâmetros de valores.
2. Use a `GetImageSetMetadata`ação com os valores dos `imageSetId` parâmetros `datastoreId` e para recuperar os valores de metadados associados para `studyInstanceUIDseriesInstanceUID`, e. `sopInstanceUID` Para obter mais informações, consulte Obtendo metadados do conjunto de imagens.
3. Crie uma URL para a solicitação usando os valores para `datastoreId` `studyInstanceUIDseriesInstanceUID`,`sopInstanceUID`,, `imageSetId` e. Para ver todo o caminho do URL no exemplo a seguir, role até o botão Copiar. O URL tem o formato:

   ```
   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. Prepare e envie sua solicitação. `GetDICOMInstance`usa uma solicitação HTTP GET com o protocolo de AWS assinatura Signature versão 4. O exemplo de código a seguir usa a ferramenta de linha de `curl` comando para obter uma instância DICOM (`.dcm`arquivo) de HealthImaging.

   Shell

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

> **Note**
>
> O `transfer-syntax` UID é opcional e o padrão é Explicit VR Little Endian se não estiver incluído. As sintaxes de transferência suportadas incluem:
>
> - Explicit VR Little Endian (ELE) - `1.2.840.10008.1.2.1` (padrão para quadros de imagem sem perdas)
> - `transfer-syntax=*`Nesse caso, os quadros da imagem serão retornados na sintaxe de transferência armazenada.
> - JPEG 2000 de alto rendimento com opções de compressão de imagem de RPCL (somente sem perdas) - `1.2.840.10008.1.2.4.202` - se a instância estiver armazenada como HealthImaging `1.2.840.10008.1.2.4.202`
> - JPEG 2000 Lossless - `1.2.840.10008.1.2.4.90` - se a instância for armazenada como sem perdas. HealthImaging
> - Linha de base JPEG (Processo 1): sintaxe de transferência padrão para compactação de imagem JPEG de 8 bits com perdas - `1.2.840.10008.1.2.4.50` - se a instância estiver armazenada em como HealthImaging `1.2.840.10008.1.2.4.50`
> - Compressão de imagem JPEG 2000 - `1.2.840.10008.1.2.4.91` - se a instância estiver armazenada em HealthImaging como `1.2.840.10008.1.2.4.91`
> - Compressão de imagem JPEG 2000 de alto rendimento - `1.2.840.10008.1.2.4.203` - se a instância estiver armazenada em como HealthImaging `1.2.840.10008.1.2.4.203`
> - Compressão de imagem JPEG XL - `1.2.840.10008.1.2.4.112` - se a instância estiver armazenada em como HealthImaging `1.2.840.10008.1.2.4.112`
> - Instâncias armazenadas HealthImaging com um ou mais quadros de imagem codificados na família MPEG de sintaxes de transferência (que inclui MPEG-4 AVC/H.264 and HEVC/H .265) podem ser MPEG2 recuperadas com o UID de sintaxe de transferência correspondente. Por exemplo, `1.2.840.10008.1.2.4.100` se a instância for armazenada como Nível MPEG2 principal do perfil principal.

Para obter mais informações, consulte Sintaxes de transferência compatíveis e Bibliotecas de decodificação de quadros de imagem para AWS HealthImaging.

Obtendo metadados da instância DICOM de HealthImaging

Use a `GetDICOMInstanceMetadata` ação para recuperar os metadados de uma instância DICOM em um armazenamento de HealthImaging dados especificando a série, o estudo e a instância UIDs associados ao recurso. A API só retornará metadados de instância dos conjuntos de imagens primários, a menos que o parâmetro opcional do conjunto de imagens seja fornecido. Você pode recuperar qualquer metadado de instância (de conjuntos de imagens primárias ou não primárias) no armazenamento de dados especificando o `imageSetId` como um parâmetro de consulta.

Para obter metadados da instância DICOM () **`.json`**

1. Colete HealthImaging `datastoreId` e defina `imageSetId` parâmetros de valores.
2. Use a `GetImageSetMetadata`ação com os valores dos `imageSetId` parâmetros `datastoreId` e para recuperar os valores de metadados associados para `studyInstanceUIDseriesInstanceUID`, e. `sopInstanceUID` Para obter mais informações, consulte Obtendo metadados do conjunto de imagens.
3. Crie uma URL para a solicitação usando os valores para `datastoreId` `studyInstanceUIDseriesInstanceUID`,`sopInstanceUID`,, `imageSetId` e. Para ver todo o caminho do URL no exemplo a seguir, role até o botão Copiar. O URL tem o formato:

   ```
   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. Prepare e envie sua solicitação. `GetDICOMInstanceMetadata`usa uma solicitação HTTP GET com o protocolo de AWS assinatura Signature versão 4. O exemplo de código a seguir usa a ferramenta de linha de `curl` comando para obter metadados (`.json`arquivo) da instância DICOM. HealthImaging

   Shell

   ```
   curl --request GET \
     'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
   d9a2a515ab294163a2d2f4069eed584c/
   studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
   series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/
   instances/1.3.6.1.4.1.5962.1.1.4.1.1.20040826186059.5457/metadata?
   imageSetId=459e50687f121185f747b67bb60d1bc8' \
     --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
   ```

```
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom+json'
```

> **Note**
>
> O UID da sintaxe de transferência indicado nos metadados corresponde ao UID da sintaxe de transferência armazenado () em. `StoredTransferSyntaxUID` HealthImaging

Obtendo metadados da série DICOM de HealthImaging

Use a `GetDICOMSeriesMetadata` ação para recuperar os metadados de uma série DICOM (`.json`arquivo) de um HealthImaging armazenamento de dados. Você pode recuperar metadados da série para qualquer conjunto de imagens primárias no armazenamento de HealthImaging dados especificando o estudo e a série UIDs associados ao recurso. Você pode recuperar metadados da série para conjuntos de imagens não primárias fornecendo o ID do conjunto de imagens como um parâmetro de consulta. Os metadados da série são retornados em `DICOM JSON` formato.

Para obter metadados da série DICOM () **`.json`**

1. Colete HealthImaging `datastoreId` e defina `imageSetId` parâmetros de valores.
2. Crie uma URL para a solicitação usando os valores para`datastoreId`, `studyInstanceUID`, `seriesInstanceUID`, e opcionalmente`imageSetId`. Para ver todo o caminho do URL no exemplo a seguir, role até o botão Copiar. O URL tem o formato:

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

3. Prepare e envie sua solicitação. `GetDICOMSeriesMetadata`usa uma solicitação HTTP GET com o protocolo de AWS assinatura Signature versão 4. O exemplo de código a seguir usa a ferramenta de linha de `curl` comando para obter metadados (`.json`arquivo) de HealthImaging.

   Shell

   ```
   curl --request GET \
   ```

```
 'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
d9a2a515ab294163a2d2f4069eed584c/
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/metadata \
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom+json' \
  --output 'series-metadata.json'
```

Com o `imageSetId` parâmetro opcional.

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**
>
> - O `imageSetId` parâmetro é necessário para recuperar metadados da série para conjuntos de imagens não primárias. A `GetDICOMInstanceMetadata` ação só retornará metadados de série para conjuntos de imagens primárias se`datastoreId`,`studyInstanceUID`, `seriesInstanceUID` forem especificados (sem um`imagesetID`).

Obtendo quadros de instância DICOM de HealthImaging

Use a `GetDICOMInstanceFrames` ação para recuperar quadros de imagem individuais ou em lote (`multipart`solicitação) de uma instância DICOM em um armazenamento de HealthImaging dados especificando o UID da série, o UID do estudo, a instância e os números dos quadros UIDs associados a um recurso. Você pode especificar o conjunto de imagens do qual os quadros de instância devem ser recuperados fornecendo o ID do conjunto de imagens como um parâmetro de consulta. A API retornará somente quadros de instância dos conjuntos de imagens primários, a menos que o parâmetro opcional do conjunto de imagens seja fornecido. Você pode recuperar qualquer quadro de instância (de conjuntos de imagens primárias ou não primárias) no armazenamento de dados especificando o `imageSetId` como um parâmetro de consulta.

Os dados DICOM podem ser recuperados em sua sintaxe de transferência armazenada ou no formato não compactado (ELE).

Para obter quadros de instância DICOM () **`multipart`**

1. Colete HealthImaging `datastoreId` e defina `imageSetId` parâmetros de valores.
2. Use a `GetImageSetMetadata`ação com os valores dos `imageSetId` parâmetros `datastoreId` e para recuperar os valores de metadados associados para `studyInstanceUIDseriesInstanceUID`, e. `sopInstanceUID` Para obter mais informações, consulte Obtendo metadados do conjunto de imagens.
3. Determine os quadros de imagem a serem recuperados dos metadados associados para formar o `frameList` parâmetro. O `frameList` parâmetro é uma lista separada por vírgula de um ou mais números de quadros não duplicados, em qualquer ordem. Por exemplo, o primeiro quadro de imagem nos metadados será o quadro 1.
   - Solicitação de quadro único: `/frames/1`
   - Solicitação de vários quadros: `/frames/1,2,3,4`
4. Crie uma URL para a solicitação usando os valores para `datastoreId` `studyInstanceUIDseriesInstanceUID`,`sopInstanceUID`,`imageSetId`,, `frameList` e. Para ver todo o caminho do URL no exemplo a seguir, role até o botão Copiar. O URL tem o formato:

```
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. Prepare e envie sua solicitação. `GetDICOMInstanceFrames`usa uma solicitação HTTP GET com o protocolo de AWS assinatura Signature versão 4. O exemplo de código a seguir usa a ferramenta de linha de `curl` comando para obter quadros de imagem em uma `multipart` resposta de HealthImaging.

   Shell

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

> **Note**
>
> O `transfer-syntax` UID é opcional e o padrão é Explicit VR Little Endian se não estiver incluído. Se a transcodificação para ELE não for viável (devido à importação com aviso), os pixels serão retornados sem transcodificação. As sintaxes de transferência suportadas incluem:
>
> - Explicit VR Little Endian (ELE) - `1.2.840.10008.1.2.1` (padrão para quadros de imagem sem perdas)
> - `transfer-syntax=*`Nesse caso, os quadros da imagem serão retornados na sintaxe de transferência armazenada.
> - JPEG 2000 de alto rendimento com opções de compressão de imagem de RPCL (somente sem perdas) - `1.2.840.10008.1.2.4.202` - se a instância estiver armazenada como HealthImaging `1.2.840.10008.1.2.4.202`
> - JPEG 2000 Lossless - `1.2.840.10008.1.2.4.90` - se a instância for armazenada como sem perdas. HealthImaging

- Linha de base JPEG (Processo 1): sintaxe de transferência padrão para compactação de imagem JPEG de 8 bits com perdas - `1.2.840.10008.1.2.4.50` - se a instância estiver armazenada em como HealthImaging `1.2.840.10008.1.2.4.50`
- Compressão de imagem JPEG 2000 - `1.2.840.10008.1.2.4.91` - se a instância estiver armazenada em HealthImaging como `1.2.840.10008.1.2.4.91`
- Compressão de imagem JPEG 2000 de alto rendimento - `1.2.840.10008.1.2.4.203` - se a instância estiver armazenada em como HealthImaging `1.2.840.10008.1.2.4.203`
- Compressão de imagem JPEG XL - `1.2.840.10008.1.2.4.112` - se a instância estiver armazenada em como HealthImaging `1.2.840.10008.1.2.4.112`
- Instâncias armazenadas HealthImaging com um ou mais quadros de imagem codificados na família MPEG de sintaxes de transferência (que inclui MPEG-4 AVC/H.264 and HEVC/H .265) podem ser MPEG2 recuperadas com o UID de sintaxe de transferência correspondente. Por exemplo, `1.2.840.10008.1.2.4.100` se a instância for armazenada como Nível MPEG2 principal do perfil principal.
- Você pode receber um 406 `NotAcceptableException` se a sintaxe de transferência solicitada não puder ser retornada com base na sintaxe de transferência armazenada ou se houver avisos de processamento específicos para a instância. Se isso ocorrer, repita a chamada com`transfer-syntax=*`.

Para obter mais informações, consulte Sintaxes de transferência compatíveis e Bibliotecas de decodificação de quadros de imagem para AWS HealthImaging.

Obtendo dados em massa DICOM de HealthImaging

Use a `GetDICOMBulkdata` ação para recuperar dados binários que foram separados dos metadados DICOM em um HealthImaging armazenamento de dados. Ao recuperar metadados de instâncias ou séries, atributos binários maiores que 1 MB serão representados por um `BulkDataURI` em vez de valores embutidos. Você pode recuperar os dados binários de qualquer conjunto de imagens primárias no armazenamento de HealthImaging dados usando o `BulkDataURI` fornecido na resposta de metadados. Você pode recuperar dados em massa para conjuntos de imagens não primárias fornecendo o ID do conjunto de imagens como um parâmetro de consulta.

Para obter dados em massa do DICOM

Quando você recupera metadados DICOM de uma ação do HealthImaging DICOMweb WADO-RS, como `GetDICOMInstanceMetadata` ou`GetDICOMSeriesMetadata`, atributos binários grandes serão substituídos em linha por, conforme mostrado abaixo: BulkData URIs

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

Para recuperar um elemento DICOM com a `GetDICOMBulkdata` ação, use as etapas a seguir.

1. Crie uma URL para a solicitação usando os valores do`BulkDataURI`, do formato:

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

2. Emita seu `GetDICOMBulkdata` comando como uma solicitação HTTP GET com o protocolo de assinatura AWS Signature versão 4. O exemplo de código a seguir usa a ferramenta de linha de `curl` comando para recuperar um elemento DICOM de um conjunto de imagens primário:

   ```
   curl --request GET \
     'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
   d9a2a515ab294163a2d2f4069eed584c/
   studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
   series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/
   instances/1.2.840.10008.5.1.4.1.1.7/bulkdata/b026324c6904b2a9cb4b88d6d61c81d1' \
     --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
     --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
     --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
     --header 'Accept: application/octet-stream' \
     --output 'bulkdata.bin'
   ```

   Para recuperar um elemento de dados DICOM de um conjunto de imagens não primário, forneça um parâmetro: `ImageSetId`

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

```
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/
instances/1.2.840.10008.5.1.4.1.1.7/bulkdata/b026324c6904b2a9cb4b88d6d61c81d1?
imageSetId=459e50687f121185f747b67bb60d1bc8' \
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/octet-stream' \
  --output 'bulkdata.bin'
```

> **Note**
>
> O `imageSetId` parâmetro é necessário para recuperar dados em massa para conjuntos de imagens não primárias. A DICOMBulkdata ação Obter só retornará dados em massa para conjuntos de imagens primárias se`datastoreId`, `studyInstanceUID`, `seriesInstanceUID`, e `SOPInstanceUID` forem especificados (sem um`imagesetID`).

Pesquisando dados DICOM em HealthImaging

HealthImaging A AWS oferece representações do DICOMweb QIDO-RS APIs para pesquisar estudos, séries e instâncias por ID do paciente e receber seus identificadores exclusivos para uso posterior. HealthImagingO DICOMweb QIDO-RS da APIs oferece flexibilidade na forma como você pesquisa dados armazenados HealthImaging e fornece interoperabilidade com aplicativos legados.

> **Importante**
>
> HealthImaging's DICOMweb APIs podem ser usados para retornar informações do conjunto de imagens com o QIDO-RS. HealthImaging DICOMweb APIs referencie somente conjuntos de imagens, salvo indicação em contrário. Use ações nativas da HealthImaging nuvem ou o parâmetro opcional de DICOMweb ações do conjunto de imagens para recuperar conjuntos de imagens não primárias. HealthImaging's DICOMweb APIs podem ser usados para retornar informações do conjunto de imagens com respostas em DICOMweb conformidade com -.
>
> HealthImaging DICOMweb As ações do QIDO-RS podem retornar no máximo 10.000 registros. Caso existam mais de 10.000 recursos, eles não serão recuperáveis por meio

das ações do QIDO-RS, mas poderão ser recuperados por meio de ações do DICOMweb WADO-RS ou ações nativas da nuvem.

Os APIs itens listados nesta seção foram criados em conformidade com o padrão DICOMweb (QIDO-RS) para imagens médicas baseadas na web. Eles não são oferecidos por meio de AWS CLI AWS SDKs e.

DICOMweb pesquisar APIs por HealthImaging

A tabela a seguir descreve todas as HealthImaging representações do DICOMweb QIDO-RS APIs disponíveis para pesquisar dados em. HealthImaging

HealthImaging representações do DICOMweb QIDO-RS APIs

Name (Nome)	Description
`SearchDICOMStudies`	Pesquise estudos DICOM HealthImaging especificando elementos de consulta de pesquisa usando uma solicitação GET. Os resultados da pesquisa do estudo são retornados no formato JSON, ordenados pela última atualização, com data decrescente (da mais recente para a mais antiga). Consulte Pesquise estudos.
`SearchDICOMSeries`	Pesquise a série DICOM HealthImaging especificando os elementos da consulta de pesquisa usando uma solicitação GET. Os resultados da pesquisa em série são retornado s no formato JSON, ordenados por `Series Number (0020, 0011)` ordem crescente (do mais antigo para o mais recente). Consulte Pesquisar séries.
`SearchDICOMInstances`	Pesquise instâncias DICOM HealthImaging especificando elementos de consulta de pesquisa usando uma solicitação GET. Os resultados da pesquisa de instâncias são

Name (Nome)	Description
	retornados no formato JSON, ordenados por `Instance Number (0020, 0013)` ordem crescente (do mais antigo para o mais recente). Consulte Pesquisar instâncias.

Tipos DICOMweb de consulta compatíveis para HealthImaging

HealthImaging suporta consultas hierárquicas de recursos QIDO-RS nos níveis de estudo, série e instância SOP. Ao usar a pesquisa hierárquica QIDO-RS para: HealthImaging

- A pesquisa por estudos retorna uma lista de estudos
- Pesquisar uma série de estudos requer uma série conhecida `StudyInstanceUID` e retorna uma lista de séries
- Pesquisar uma lista de instâncias requer uma informação conhecida `StudyInstanceUID` e `SeriesInstanceUID`

A tabela a seguir descreve os tipos de consulta hierárquica QIDO-RS compatíveis para pesquisar dados em. HealthImaging

HealthImaging tipos de consulta QIDO-RS suportados

Tipo da consulta	Exemplo
Consultas de valores de atributos	Pesquise todas as séries em um estudo ondemodality=CT . `.../studies/1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894866/series?00080060=CT` Pesquise todos os estudos em que o ID do paciente e a data do estudo sejam esses valores, respectivamente.

Tipo da consulta	Exemplo
	`…/studies?PatientID=1123581 3&StudyDate=20130509`
Consultas de palavras-chave	Pesquise todas as séries usando a `SeriesIns tanceUID` palavra-chave. `.../studies/1.3.6.1.4.1.145 19.5.2.1.6279.6001.10137060 527657755614301389486 6/seri es?SeriesInstanceUID=1.3.6. 1.4.1.14519.5.2.1.6279.6001 .101370605276577556143013894868`
Consultas de tags	Pesquise tags usando parâmetros de consulta passados no group/element formulário. {group} {element} como 0020000D
Intervalo de consultas	`...?Modality=CT&StudyDate=A ABBYYYY-BBCCYYYY`
Paginação de resultados com e `limit` `offset`	`.../studies?limit=1&offset= 0&00080020=20000101` Você pode usar os parâmetros de limite e deslocamento para paginar as respostas da pesquisa. O valor padrão do limite é 1000 e veja HealthImaging Endpoints e cotas da AWS o valor máximo. Limite máximo = 1000, deslocamento máximo = 9000

Tipo da consulta	Exemplo

Consultas curingas	As consultas curinga oferecem mais flexibilidade na pesquisa usando “*” e “?”. “*” corresponde a qualquer sequência de caracteres (incluindo um valor de comprimento zero) e “?” corresponde a qualquer caractere único.
	
	Pesquise todos os estudos em um armazenamento de dados que StudyDescription contenha “Nuclear”:
	
	`.../studies?StudyDescription=*Nuclear*`
	
	Pesquise todos os estudos que StudyDescription terminam com “Nuclear”:
	
	`.../studies?StudyDescription=*Nuclear`
	
	Pesquise todos os estudos que StudyDescription começam com “Nuclear”:
	
	`.../studies?StudyDescription=Nuclear*`
	
	Pesquise todos os estudos em que o PatientID tenha exatamente 3 caracteres após 200965981:
	
	`.../studies?PatientID=200965981???`

Tipo da consulta	Exemplo
FuzzyMatching consultas	Ative a correspondência difusa nos atributos DICOM do nome ((0010.0010), PatientNa me ReferringPhysicianName (0008.0090)) adicionando o parâmetro de consulta opcional fuzzymatching: `.../studies?fuzzymatching=t rue&PatientName="Thomas^Albert"` Essa consulta realiza a correspondência de palavras de prefixo sem distinção entre maiúsculas e minúsculas em qualquer parte do valor. PatientName Ele retorna resultados com PatientName valores como “thomas”, “Albert”, “Thomas Albert”, “Thomas^Albert”, mas não “hom” ou “ber”.

Tópicos

- Pesquisando estudos DICOM em HealthImaging
- Pesquisando a série DICOM em HealthImaging
- Pesquisando instâncias DICOM em HealthImaging

Pesquisando estudos DICOM em HealthImaging

Use a `SearchDICOMStudies` API para pesquisar estudos DICOM em um armazenamento de HealthImaging dados. Você pode pesquisar estudos DICOM criando uma URL que inclua elementos de dados DICOM compatíveis (atributos). HealthImaging Os resultados da pesquisa do estudo são retornados no formato JSON, ordenados pela última atualização, com data decrescente (da mais recente para a mais antiga).

Para pesquisar estudos DICOM

1. Colecione HealthImaging `region` e `datastoreId` valorize. Para obter mais informações, consulte Obter propriedades do datastore.

2. Crie uma URL para a solicitação, incluindo todos os elementos de estudo aplicáveis. Para ver todo o caminho do URL no exemplo a seguir, role até o botão Copiar. O URL tem o formato:

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

Elementos de estudo para **`SearchDICOMStudies`**

Etiqueta de elemento DICOM	Nome do elemento DICOM
(0008,0020)	Study Date
(0008,0030)	StudyTime
(0008,0050)	Accession Number
(0008,0061)	Modalities in Study
(0008,0090)	Referring Physician Name
(0008,1030)	Study Description
(0010,0010)	Patient Name
(0010,0020)	Patient ID
(0010,0030)	Patient BirthDate
(0010,0032)	Patient BirthTime
(0020,000D)	Study Instance UID
(0020,0010)	Study ID

3. Prepare e envie sua solicitação. `SearchDICOMStudies`usa uma solicitação HTTP GET com o protocolo de AWS assinatura Signature versão 4. O exemplo a seguir usa a ferramenta de linha de `curl` comando para pesquisar informações sobre estudos DICOM.

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
```

Os resultados da pesquisa do estudo são retornados no formato JSON, ordenados pela última atualização, com data decrescente (da mais recente para a mais antiga).

Pesquisando a série DICOM em HealthImaging

Use a `SearchDICOMSeries` API para pesquisar séries DICOM em um armazenamento de HealthImaging dados. Você pode pesquisar a série DICOM criando uma URL que inclua elementos de dados DICOM compatíveis (atributos). HealthImaging Os resultados da pesquisa em série são retornados no formato JSON, ordenados em ordem crescente (do mais antigo para o mais recente).

Para pesquisar a série DICOM

1. Colecione HealthImaging `region` e `datastoreId` valorize. Para obter mais informações, consulte Obter propriedades do datastore.
2. Colete o `StudyInstanceUID` valor. Para obter mais informações, consulte Obtendo metadados do conjunto de imagens.
3. Crie uma URL para a solicitação, incluindo todos os elementos da série aplicáveis. Para ver todo o caminho do URL no exemplo a seguir, role até o botão Copiar. O URL tem o formato:

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

Elementos da série para **`SearchDICOMSeries`**

Etiqueta de elemento DICOM	Nome do elemento DICOM
`(0008,0060)`	`Modality`
`(0020,000E)`	`Series Instance UID`

4. Prepare e envie sua solicitação. `SearchDICOMSeries`usa uma solicitação HTTP GET com o protocolo de AWS assinatura Signature versão 4. O exemplo a seguir usa a ferramenta de linha de `curl` comando para pesquisar informações da série DICOM.

   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
   ```

   Os resultados da pesquisa em série são retornados no formato JSON, ordenados por `Series Number (0020,0011)` ordem crescente (do mais antigo para o mais recente).

Pesquisando instâncias DICOM em HealthImaging

Use a `SearchDICOMInstances` API para pesquisar instâncias DICOM em um armazenamento de HealthImaging dados. Você pode pesquisar instâncias DICOM criando uma URL que inclua elementos de dados DICOM compatíveis (atributos). HealthImaging Os resultados da instância são retornados no formato JSON, ordenados em ordem crescente (do mais antigo para o mais recente).

Para pesquisar instâncias DICOM

1. Colecione HealthImaging `region` e `datastoreId` valorize. Para obter mais informações, consulte Obter propriedades do datastore.
2. Colete valores para `StudyInstanceUID` `SeriesInstanceUID` e. Para obter mais informações, consulte Obtendo metadados do conjunto de imagens.
3. Crie uma URL para a solicitação, incluindo todos os elementos de pesquisa aplicáveis. Para ver todo o caminho do URL no exemplo a seguir, role até o botão Copiar. O URL tem o formato:

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

Elementos de instância para **`SearchDICOMInstances`**

Etiqueta de elemento DICOM	Nome do elemento DICOM
`(0008,0016)`	`SOP Class UID`
`(0008,0018)`	`SOP Instance UID`
`(0008,1196)`	`WarningReason`

HealthImaging usa o elemento DICOM (0008,1196) para manter os códigos de aviso de importação. Os códigos de aviso de importação podem ser pesquisados no nível da instância. Os códigos de aviso de importação podem ser pesquisados com caracteres curinga ou códigos de aviso específicos. Consulte HealthImaging Códigos de aviso.

4. Prepare e envie sua solicitação. `SearchDICOMInstances`usa uma solicitação HTTP GET com o protocolo de AWS assinatura Signature versão 4. O exemplo a seguir usa a ferramenta de linha de `curl` comando para pesquisar informações sobre instâncias DICOM.

   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
   ```

   Os resultados da pesquisa de instâncias são retornados no formato JSON, ordenados por `Instance Number (0020,0013)` ordem crescente (do mais antigo para o mais recente)

Autenticação OIDC para DICOMweb APIs

A AWS HealthImaging oferece suporte à autenticação baseada em OAuth 2.0 para solicitações de DICOMweb API usando o OpenID Connect (OIDC), além da autenticação AWS Signature Version 4 (SigV4) existente. O OIDC permite que você se integre HealthImaging diretamente com provedores

de identidade externos (IdPs) e permite que você forneça aos aplicativos baseados em padrões acesso aos seus dados de imagens médicas por meio de HealthImaging DICOMweb terminais, sem exigir que cada aplicativo tenha credenciais. AWS

Tópicos

- Verificação personalizada de tokens com autorizadores Lambda
- Configurar um autorizador AWS Lambda para autenticação OIDC

Verificação personalizada de tokens com autorizadores Lambda

HealthImaging implementa o suporte do OIDC por meio de uma arquitetura que usa autorizadores Lambda, permitindo que os clientes implementem sua própria lógica de verificação de token. Esse design oferece controle flexível sobre como os tokens são validados e como as decisões de acesso são aplicadas, acomodando um cenário diversificado de provedores de identidade compatíveis com OIDC (IdPs) e vários métodos de verificação de tokens.

Fluxo de autenticação

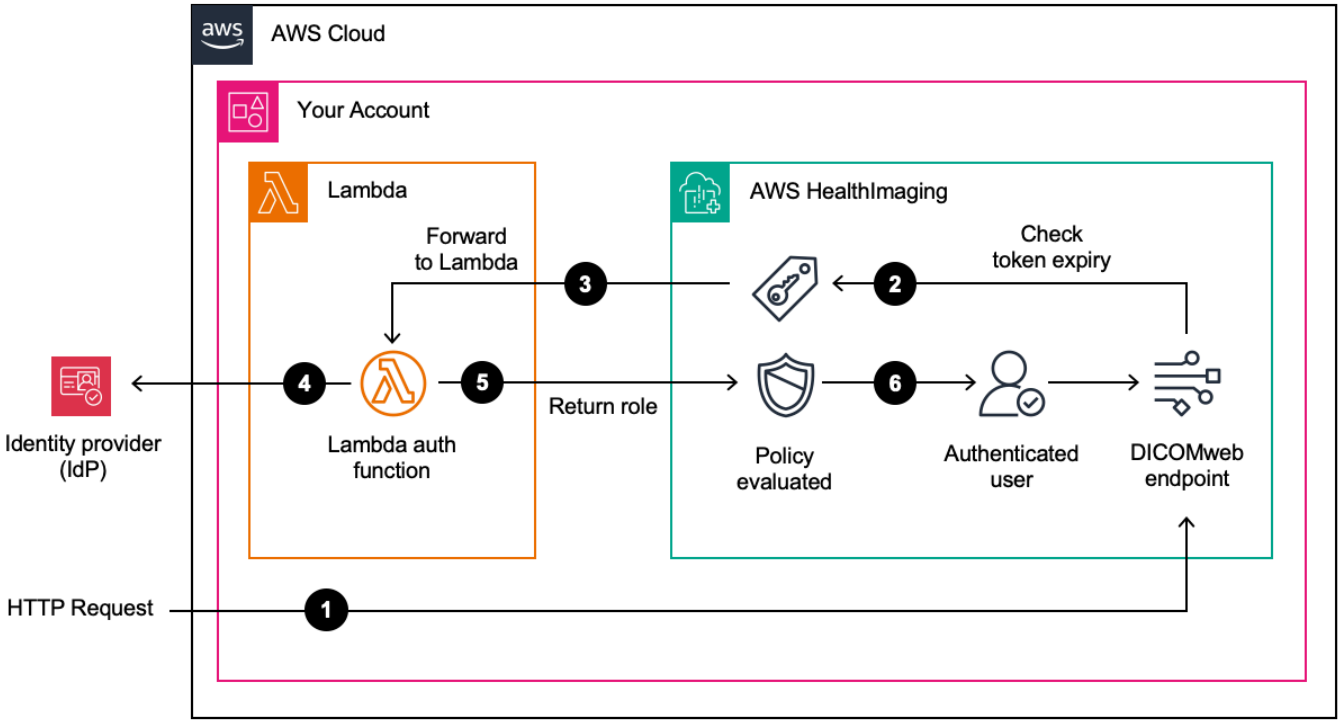
Veja como a autenticação funciona em alto nível:

1. O cliente chama a DICOMweb API: seu aplicativo se autentica com o provedor de identidade OIDC escolhido e recebe um token de ID assinado (JWT). Para cada solicitação DICOMweb HTTP, o cliente deve incluir o token de acesso OIDC no cabeçalho de autorização (normalmente um token de portador). Antes que a solicitação chegue aos seus dados, HealthImaging extrai esse token da solicitação recebida e chama um autorizador Lambda que você configura.
   a. O cabeçalho normalmente segue o formato:`Authorization: Bearer <token>`.
2. Verificação inicial: HealthImaging verifica as reivindicações do token de acesso para rejeitar rapidamente quaisquer tokens obviamente inválidos ou expirados sem invocar a função Lambda desnecessariamente. HealthImaging executa uma verificação inicial de determinadas declarações padrão no token de acesso antes de invocar o autorizador Lambda:
   a. `iat`(Emitido em): HealthImaging verifica se o tempo de emissão do token está dentro dos limites aceitáveis.
   b. `exp`(Tempo de expiração): HealthImaging verifica se o token não expirou.
   c. `nbf`(Não antes da hora): Se presente, HealthImaging garante que o token não esteja sendo usado antes do horário de início válido.

3. HealthImaging invoca um autorizador do Lambda: se a verificação inicial da solicitação for aprovada, HealthImaging delegará a verificação adicional do token à função autorizadora do Lambda configurada pelo cliente. HealthImaging passa o token extraído e outras informações relevantes da solicitação para a função Lambda. A função Lambda verifica a assinatura e as declarações do token.
4. Verifique com um provedor de identidade: o Lambda contém um código personalizado que verifica a assinatura do token de ID, realiza uma verificação mais abrangente do token (por exemplo, emissor, público, declarações personalizadas) e valida essas declarações em relação ao IdP quando necessário.
5. O autorizador retorna uma política de acesso: após a verificação bem-sucedida, a função Lambda determina as permissões apropriadas para o uso autenticado. Em seguida, o autorizador do Lambda retorna o nome do recurso da Amazon (ARN) de uma função do IAM que representa o conjunto de permissões a serem concedidas.
6. Execução da solicitação: se a função assumida do IAM tiver as permissões necessárias, HealthImaging continue com a devolução do DICOMWeb recurso solicitado. Se as permissões forem insuficientes, HealthImaging nega a solicitação e retorna um erro de resposta de erro apropriado (ou seja, 403 Forbidden).

> **Note**
>
> A função lambda do autorizador não é gerenciada pelo serviço da AWS HealthImaging . Ele é executado em sua AWS conta. Os clientes são cobrados pela invocação da função e pelo tempo de execução separadamente de suas HealthImaging cobranças.

Visão geral da arquitetura



Fluxo de trabalho de autenticação OIDC com o autorizador Lambda

Pré-requisitos

Requisitos de token de acesso

HealthImaging exige que o token de acesso esteja no formato JSON Web Token (JWT). Muitos provedores de identidade (IDPs) oferecem esse formato de token nativamente, enquanto outros permitem que você selecione ou configure o formulário do token de acesso. Certifique-se de que seu IDP escolhido possa emitir tokens JWT antes de prosseguir com a integração.

Formato do token

O token de acesso deve estar no formato JWT (JSON Web Token)

Declarações necessárias

`exp`(Tempo de expiração)

Reivindicação obrigatória que especifica quando o token se torna inválido.

- Deve ser posterior à hora atual em UTC
- Representa quando o token se torna inválido

`iat`(Emitido em)

Reivindicação obrigatória que especifica quando o token foi emitido.

- Deve ser antes da hora atual em UTC
- NÃO deve ser antes de 12 horas antes da hora atual em UTC
- Isso efetivamente impõe uma vida útil máxima do token de 12 horas.

`nbf`(Não antes do tempo)

Declaração opcional que especifica a primeira vez em que o token pode ser usado.

- Se presente, será avaliado por HealthImaging
- Especifica o tempo antes do qual o token não deve ser aceito

Requisitos de tempo de resposta do autorizador Lambda

HealthImaging impõe requisitos rígidos de tempo para as respostas do autorizador Lambda para garantir o desempenho ideal da API. Sua função Lambda deve retornar em 1 segundo.

Práticas recomendadas

Otimize a verificação de tokens

- Armazene JWKS (JSON Web Key Sets) quando possível
- Armazene tokens de acesso válidos em cache quando possível
- Minimize as chamadas de rede para seu provedor de identidade
- Implemente uma lógica eficiente de validação de tokens

Configuração Lambda

- As funções baseadas em Python e Node.js normalmente inicializam mais rápido
- Reduza a quantidade de bibliotecas externas a serem carregadas
- Configure a alocação de memória adequada para garantir um desempenho consistente
- Monitore os tempos de execução usando CloudWatch métricas

Ativação da autenticação OIDC

- A autenticação OIDC só pode ser ativada ao criar um novo armazenamento de dados
- A ativação do OIDC para armazenamentos de dados existentes não é suportada pela API
- Para habilitar o OIDC em um armazenamento de dados existente, os clientes devem entrar em contato com o Support AWS

Configurar um autorizador AWS Lambda para autenticação OIDC

Este guia pressupõe que você já tenha configurado o provedor de identidade (IdP) de sua escolha para fornecer tokens de acesso compatíveis com os requisitos do recurso de autenticação HealthImaging do OIDC.

1. Configurar funções do IAM para acesso à DICOMWeb API

Antes de configurar o autorizador Lambda, crie funções do IAM HealthImaging para assumir ao DICOMWeb processar solicitações de API. A função autorizadora Lambda retorna uma dessas funções ARN após a verificação bem-sucedida do token, HealthImaging permitindo executar as solicitações com as permissões apropriadas.

1. Crie políticas do IAM definindo os privilégios de DICOMWeb API desejados. Consulte a seção "Usando DICOMweb" da HealthImaging documentação para ver as permissões disponíveis.
2. Crie funções do IAM que:
   - Anexe essas políticas
   - Inclua uma relação de confiança que permita que o diretor HealthImaging de serviço da AWS (`medical-imaging.amazonaws.com`) assuma essas funções.

Aqui está um exemplo de uma política que permite que funções associadas acessem a API HealthImaging DICOMWeb somente para leitura:

JSON

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

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

Aqui está um exemplo da política de relacionamento de confiança que deve ser associada à (s) função (s):

JSON

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

O autorizador Lambda que você criará na próxima etapa pode avaliar as reivindicações de token e retornar o ARN da função apropriada. A AWS então HealthImaging representará essa função para executar a solicitação de DICOMWeb API com as permissões correspondentes.

Por exemplo:

- Um token com declarações de “administrador” pode retornar um ARN para uma função com acesso total
- Um token com declarações de “leitor” pode retornar um ARN para uma função com acesso somente de leitura
- Um token com declarações “Department_A” pode retornar um ARN para uma função específica do nível de acesso desse departamento

Esse mecanismo permite que você mapeie o modelo de autorização do seu IdP para HealthImaging permissões específicas da AWS por meio de funções do IAM.

2. Criar e configurar a função Lambda Authorizer

Crie uma função Lambda que verificará o token JWT e retornará o ARN apropriado da função do IAM com base na avaliação das declarações do token. Essa função é invocada pelo serviço de imagens de saúde e transmitida por um evento que contém o ID do HealthImaging armazenamento de dados, a DICOMWeb operação e o token de acesso encontrados na solicitação HTTP:

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

A função autorizadora do Lambda deve retornar uma resposta JSON com a seguinte estrutura:

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

Você pode consultar o exemplo de implementação para obter mais informações.

> **Note**
>
> Como a DICOMWeb solicitação só é respondida depois que o token de acesso é verificado pelo autorizador lambda, é importante que a execução dessa função seja a mais rápida possível para fornecer o melhor tempo de resposta da DICOMWeb API.

Para que o HealthImaging serviço seja autorizado a invocar a função autorizadora lambda, ele deve ter uma política de recursos que permita que o HealthImaging serviço a invoque. Essa política de recursos pode ser criada no menu de permissões da guia de configuração lambda ou usando AWS CLI:

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

Essa política de recursos permite que o HealthImaging serviço invoque seu autorizador Lambda ao DICOMWeb autenticar solicitações de API.

> **Note**
>
> A política de recursos lambda pode ser atualizada posteriormente com uma condição "ArnLike" correspondente ao ARN de um HealthImaging armazenamento de dados específico.

Aqui está um exemplo da política de recursos lambda:

JSON

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "LambaAuthorizer-HealthImagingInvokePermission",
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "medical-imaging.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:123456789012::function:
{LambdaAuthorizerFunctionName}",
      "Condition": {
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/datastore-123"
        }
      }
    }
  ]
}
```

3. Crie um novo armazenamento de dados com a autenticação OIDC

Para habilitar a autenticação OIDC, você deve criar um novo armazenamento de dados usando o AWS CLI com o parâmetro "". lambda-authorizer-arn A autenticação OIDC não pode ser habilitada em datastores existentes sem entrar em contato com o Support. AWS

Veja um exemplo de como criar um novo armazenamento de dados com a autenticação OIDC ativada:

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

Você pode verificar se um armazenamento de dados específico tem o recurso de autenticação OIDC ativado usando o comando AWS CLI get-datastore e verificando se o atributo "" está presente: lambdaAuthorizerArn

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

```
{
    "datastoreProperties": {
        "datastoreId": YourdatastoreId,
        "datastoreName": YourDatastoreName,
        "datastoreStatus": "ACTIVE",
```

```
        "lambdaAuthorizerArn": YourAuthorizerFunctionArn,
        "datastoreArn": YourDatastoreArn,
        "createdAt": "2025-09-30T14:16:04.015000-05:00",
        "updatedAt": "2025-09-30T14:16:04.015000-05:00"
    }
}
```

> **Note**
>
> A função de execução do comando de criação do AWS CLI armazenamento de dados deve ter as permissões apropriadas para invocar a função autorizadora do Lambda. Isso atenua os ataques de escalonamento de privilégios em que usuários mal-intencionados podem executar funções não autorizadas do Lambda por meio da configuração do autorizador do armazenamento de dados.

Códigos de exceção

Em caso de falha na autenticação, HealthImaging retorna os seguintes códigos de resposta de erro HTTP e mensagens corporais:

Condição	Resposta AHI
O Lambda Authorizer não existe ou é inválido	424 Configuração incorreta do autorizador
Autorizador encerrado devido a falha na execução	424 Falha no autorizador
Qualquer outro erro não mapeado do autorizador	424 Falha no autorizador
O autorizador retornou uma resposta inválida/mal formada	424 Configuração incorreta do autorizador
O autorizador foi executado por mais de 1s	Tempo limite do autorizador 408

Condição	Resposta AHI
O token está expirado ou é inválido	403 Token inválido ou expirado
O AHI não pode federar a função IAM retornada devido à configuração incorreta do autorizador	424 Configuração incorreta do autorizador
O autorizador retornou uma função vazia	403 Acesso negado
A função retornada não pode ser chamada (configuração incorreta de assume-role/trust)	424 Configuração incorreta do autorizador
A taxa de solicitação excede os limites do DICOMweb gateway	429 Muitas solicitações
Armazenamento de dados, função de retorno ou autorizador entre regiões Account/Cross	424 Autorizador: Acesso entre regiões Account/Cross

Exemplo de implementação

Este exemplo em Python demonstra uma função autorizadora lambda que verifica os tokens de acesso do Cognito a partir de eventos HealthImaging e retorna um AWS ARN de função do IAM com os privilégios apropriados. DICOMWeb

O autorizador Lambda implementa dois mecanismos de armazenamento em cache para reduzir as chamadas externas e a latência de resposta. O JWKS (JSON Web Key Set) é obtido uma vez a cada hora e armazenado na pasta temporária da função, permitindo que as invocações subsequentes da função o leiam localmente em vez de buscá-lo na rede pública. Você também notará que um objeto de dicionário token_cache é instanciado no contexto global dessa função Lambda. As variáveis globais são compartilhadas por todas as invocações que reutilizam o mesmo contexto aquecido do

Lambda. Graças a isso, os tokens verificados com sucesso podem ser armazenados neste dicionário e pesquisados rapidamente durante a próxima execução dessa mesma função Lambda. O método de armazenamento em cache representa uma abordagem generalista que pode caber em tokens de acesso emitidos pela maioria dos provedores de identidade. Para uma opção de armazenamento em cache específica do AWS Cognito, consulte a seção Gerenciando grupos de usuários e a seção de armazenamento em cache da documentação do Cognito.AWS

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

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

# Global token cache with TTL
token_cache = {}

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

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

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

def get_cached_jwks():
    """Get JWKS from cache file if valid, otherwise return None """
```

```
    try:
        if os.path.exists(JWKS_CACHE_FILE):
            # Check if cache file is still valid
            cache_age = time.time() - os.path.getmtime(JWKS_CACHE_FILE)
            if cache_age < JWKS_CACHE_TTL:
                with open(JWKS_CACHE_FILE, 'r') as f:
                    jwks = json.load(f)
                    logger.debug(f'Using cached JWKS (age: {int(cache_age)}s)')
                    return jwks
            else:
                logger.debug(f'JWKS cache expired (age: {int(cache_age)}s)')
    except Exception as e:
        logger.debug(f'Error reading JWKS cache: {e}')

    return None

def cache_jwks(jwks):
    """Cache JWKS to file"""
    try:
        with open(JWKS_CACHE_FILE, 'w') as f:
            json.dump(jwks, f)
        logger.debug('JWKS cached successfully')
    except Exception as e:
        logger.debug(f'Error caching JWKS: {e}')

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

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

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

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

```
    return cached

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

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

def handler(event, context):
    cleanup_expired_tokens() # start be removing expired tokens from the cache
    try:
        # Extract token from bearerToken or authorizationToken field
        token = event.get('bearerToken')
        if not token:
            raise Exception('No token provided')

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

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

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

        # Decode token header to get kid
        headers = jwt.get_unverified_headers(token)
```

```
        kid = headers['kid']

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

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

        if not key:
            raise Exception('Public key not found')

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

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

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

        # Cache the validated token
        cache_token(token, payload)

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

```
    except ExpiredSignatureError:
        logger.debug('Token expired')
        return {
            'isTokenValid': False,
            'roleArn': ''
        }
    except JWTClaimsError:
        logger.debug('Invalid token claims')
        return {
            'isTokenValid': False,
            'roleArn': ''
        }
    except JWTError as e:
        logger.debug('JWT validation error: %s', e)
        return {
            'isTokenValid': False,
            'roleArn': ''
        }
    except Exception as e:
        logger.debug('Authorization failed: %s', e)
        return {
            'isTokenValid': False,
            'roleArn': ''
        }
```

Importação de dados de imagem com a AWS HealthImaging

Importar é o processo de mover seus dados de imagens médicas de um bucket de entrada do Amazon S3 para um armazenamento de dados da HealthImaging AWS. Durante a importação, HealthImaging a AWS executa uma verificação de dados de pixels antes de transformar seus arquivos DICOM P10 em conjuntos de imagens compostos por metadados e quadros de imagem (dados de pixels).

> **Importante**
>
> HealthImaging os trabalhos de importação processam binários de instâncias DICOM (`.dcm`arquivos) e os transformam em conjuntos de imagens. Use ações nativas da HealthImaging nuvem (APIs) para gerenciar armazenamentos de dados e conjuntos de imagens. Use HealthImaging a representação dos DICOMweb serviços para retornar DICOMweb respostas.

Os tópicos a seguir descrevem como importar seus dados de imagens médicas para um armazenamento de HealthImaging dados usando o Console de gerenciamento da AWS AWS CLI, AWS SDKs e.

Tópicos

- Noções básicas sobre as tarefas de importação
- Como iniciar trabalho de trabalho de trabalho de trabalho
- Como obter propriedades do trabalho de importação
- Listar trabalhos de importação

Noções básicas sobre as tarefas de importação

Depois de criar um armazenamento de dados na AWS HealthImaging, você deve importar seus dados de imagens médicas do seu bucket de entrada do Amazon S3 para o seu armazenamento de dados para criar conjuntos de imagens. Você pode usar o Console de gerenciamento da AWS, AWS CLI, e AWS SDKs para iniciar, descrever e listar trabalhos de importação.

Quando você importa seus dados DICOM P10 para um datastore da AWS HealthImaging , o serviço tenta organizar automaticamente as instâncias de acordo com a hierarquia DICOM de

UID de estudo, UID de série e UID de instância, com base nos elementos de metadados. Os dados importados se tornarão primários se os elementos de metadados dos dados importados não entrarem em conflito com os conjuntos de imagens primárias existentes no armazenamento de dados. Se os elementos de metadados dos dados DICOM P10 recém-importados entrarem em conflito com os conjuntos de imagens primárias existentes, os novos dados serão adicionados aos conjuntos de imagens não primárias. Quando as importações de dados criam conjuntos de imagens não primários, a AWS HealthImaging emite um EventBridge evento com`isPrimary: False`, e o registro gravado no também `success.ndjson` estará `isPrimary: False` dentro do `importResponse` objeto.

Ao importar dados, HealthImaging faça o seguinte:

- Se as instâncias que compõem uma série DICOM forem importadas em uma tarefa de importação e não entrarem em conflito com as instâncias que já estão no armazenamento de dados, todas as instâncias serão organizadas em um conjunto de imagens primário.
- Se as instâncias que compõem uma série DICOM forem importadas em duas ou mais tarefas de importação e não entrarem em conflito com as instâncias que já estão no armazenamento de dados, todas as instâncias serão organizadas como um conjunto de imagens primárias.
- Se uma instância for importada mais de uma vez, a versão mais recente substituirá qualquer versão mais antiga armazenada em um conjunto de imagens primário, e o número da versão do conjunto de imagens primário será incrementado.

Você pode atualizar as instâncias no primário com as etapas descritas em Atualização dos metadados do conjunto de imagens.

Durante a importação, valores binários em tags privadas (com tipos de VR OB, OD, OF, OL, OV, OW, UN) que excedem 1 MB de tamanho são armazenados separadamente dos metadados. Ao recuperar metadados para essas instâncias usando `GetDICOMInstanceMetadata` ou`GetDICOMSeriesMetadata`, esses grandes valores binários são substituídos por BulkDataURIs, e os dados binários reais podem ser recuperados usando a API. `GetDICOMBulkdata`

Lembre-se dos seguintes pontos ao importar seus arquivos de imagens médicas do Amazon S3 para HealthImaging um armazenamento de dados:

- As instâncias correspondentes a uma série DICOM serão combinadas automaticamente em um único conjunto de imagens, denotado como primário.

- Você pode importar dados DICOM P10 em uma tarefa de importação ou em várias tarefas de importação, e o serviço organizará as instâncias em conjuntos de imagens primárias que correspondem à série DICOM
- As restrições de comprimento se aplicam a elementos DICOM específicos durante a importação. Para garantir uma importação bem-sucedida, verifique se os seus dados de imagens médicas não excedam as restrições de tamanho máximo. Para obter mais informações, consulte Restrições de elementos de DICOM.
- Uma verificação de dados de píxeis é realizada no início dos trabalhos de importação. Para obter mais informações, consulte Verificação de dados de pixel.
- Há endpoints, cotas e limites de limitação associados às ações de importação. HealthImaging Para obter mais informações, consulte Endpoints e cotas e Limites de controle de utilização.
- Para cada tarefa de importação, os resultados do processamento são armazenados no `outputS3Uri` local. Os resultados do processamento são organizados em um arquivo `job-output-manifest.json` e pastas `SUCCESS` e `FAILURE`.

> **Note**
>
> Você pode incluir até 10.000 pastas aninhadas em um único trabalho de importação.

- O arquivo `job-output-manifest.json` contém a saída `jobSummary` e detalhes adicionais sobre os dados processados. O seguinte exemplo de saída mostra um arquivo `job-output-manifest.json`.

```
{
"jobSummary": {
"jobId": "09876543210987654321098765432109",
        "datastoreId": "12345678901234567890123456789012",
        "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",
        "outputS3Uri": "s3://medical-imaging-output/
job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/",
        "successOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/SUCCESS/",
        "failureOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/FAILURE/",
```

```
        "warningsOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/WARNING/",
        "numberOfScannedFiles": 5,
        "numberOfImportedFiles": 3,
        "numberOfFilesWithCustomerError": 2,
        "numberOfFilesWithServerError": 0,
        "numberOfGeneratedImageSets": 2,
        "imageSetsSummary": [{
"imageSetId": "12345612345612345678907890789012",
                "numberOfMatchedSOPInstances": 2
            },
            {
"imageSetId": "12345612345612345678917891789012",
                "numberOfMatchedSOPInstances": 1
            }
        ]
    }
}
```

- A pasta SUCCESS contém o arquivo `success.ndjson` contendo os resultados de todos os arquivos de imagem que foram importados com sucesso. O seguinte exemplo de saída mostra um arquivo `success.ndjson`.

```
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105620.1.0.1.dcm","importResponse
{"imageSetId":"12345612345612345678907890789012", "isPrimary": True}}
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105630.1.0.1.dcm","importResponse
{"imageSetId":"12345612345612345678917891789012", "isPrimary": True}}
```

- A pasta FAILURE contém o arquivo `failure.ndjson` contendo os resultados de todos os arquivos de imagem que não foram importados com êxito. O seguinte exemplo de saída mostra um arquivo `failure.ndjson`.

```
{"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"}}
```

- A `WARNING` pasta contém o `warning.ndjson` arquivo contendo os resultados de todos os arquivos de imagem que foram importados com sucesso, mas com avisos. O seguinte exemplo de saída mostra um arquivo `warning.ndjson`.

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

- Os trabalhos de importação são mantidos na lista de trabalhos por 90 dias e depois arquivados.

Como iniciar trabalho de trabalho de trabalho de trabalho

Use a `StartDICOMImportJob` ação para iniciar uma verificação de dados em pixels e importar dados em massa para um armazenamento de HealthImaging dados da AWS. O trabalho de importação importa arquivos DICOM P10 localizados no bucket de entrada do Amazon S3 especificado pelo parâmetro `inputS3Uri`. Os resultados do processamento do trabalho de importação são armazenados no bucket de saída do Amazon S3 especificado pelo parâmetro `outputS3Uri`.

> **Note**
>
> Lembre-se dos seguintes pontos antes de iniciar um trabalho de importação:
>
> - HealthImaging suporta a importação de arquivos DICOM P10 com diferentes sintaxes de transferência. Alguns arquivos mantêm a codificação da sintaxe de transferência original durante a importação, enquanto outros são transcodificados para HTJ2 K lossless por padrão ou JPEG 2000 Lossless, dependendo da configuração do seu armazenamento de dados. Para obter mais informações, consulte Sintaxes de transferência compatíveis.
> - HealthImaging suporta importações de dados de buckets do Amazon S3 localizados em outras regiões suportadas. Para obter essa funcionalidade, forneça o `inputOwnerAccountId` parâmetro ao iniciar um trabalho de importação. Para obter mais informações, consulte Importação entre contas para AWS HealthImaging.
> - HealthImaging aplica restrições de comprimento a elementos DICOM específicos durante a importação. Para obter mais informações, consulte Restrições de elementos de DICOM.

Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `StartDICOMImportJob`a AWS HealthImaging API Reference.

Como iniciar um trabalho de importação

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.
3. Escolha Importar dados DICOM.

   A página Importar dados DICOM é aberta.
4. Na seção Detalhes, insira as seguintes informações:
   - Nome (opcional)
   - Importar localização da fonte no S3
   - ID da conta do proprietário do bucket de origem (opcional)
   - Chave de criptografia (opcional)
   - Destino de saída no S3
5. Na seção Acesso ao serviço, escolha Usar uma função de perfil de serviço existente e selecione a função no menu Nome da perfil de serviço ou escolha Criar e usar um novo perfil de serviço.
6. Escolha Importar.

AWS CLI and SDKs

C++

SDK para 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();
}
```

- Para obter detalhes sobre a API, consulte Start DICOMImport Job in AWS SDK para C++ API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para iniciar um trabalho de importação dicom

O exemplo de código `start-dicom-import-job` a seguir inicia um trabalho de importação dicom.

```
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"
```

Saída:

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

Para obter mais informações, consulte Iniciando um trabalho de importação no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte Start DICOMImport Job in AWS CLI Command Reference.

Java

SDK para 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 "";
   }
```

- Para obter detalhes sobre a API, consulte Start DICOMImport Job in AWS SDK for Java 2.x API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

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

- Para obter detalhes sobre a API, consulte Start DICOMImport Job in AWS SDK para JavaScript API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte Referência da API Start DICOMImport Job in AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_role_arn = 'arn:aws:iam::123456789012:role/ImportJobRole'
    " iv_input_s3_uri = 's3://my-bucket/input/'
    " iv_output_s3_uri = 's3://my-bucket/output/'
    oo_result = lo_mig->startdicomimportjob(
      iv_jobname = iv_job_name
      iv_datastoreid = iv_datastore_id
      iv_dataaccessrolearn = iv_role_arn
      iv_inputs3uri = iv_input_s3_uri
      iv_outputs3uri = iv_output_s3_uri ).
    DATA(lv_job_id) = oo_result->get_jobid( ).
    MESSAGE |DICOM import job started with ID: { lv_job_id }.| TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
  CATCH /aws1/cx_migservicequotaexcdex.
    MESSAGE 'Service quota exceeded.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a referência da API Start DICOMImport Job in AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Como obter propriedades do trabalho de importação

Use a `GetDICOMImportJob` ação para saber mais sobre as propriedades de trabalho HealthImaging de importação da AWS. Por exemplo, depois de iniciar um trabalho de importação, você pode executar `GetDICOMImportJob` para encontrar o status do trabalho. Quando `jobStatus` retornar como `COMPLETED`, você estará pronto para acessar seus Conjuntos de imagens.

> **Note**
>
> O `jobStatus` se refere à execução do trabalho de importação. Portanto, um trabalho de importação pode retornar `jobStatus` como `COMPLETED` mesmo que problemas de validação sejam descobertos durante o processo de importação. Se um `jobStatus` retornar como `COMPLETED`, ainda recomendamos que você revise os manifestos de saída gravados no Amazon S3, pois eles fornecem detalhes sobre o sucesso ou a falha de importações individuais de objetos P10.

Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `GetDICOMImportJob`a AWS HealthImaging API Reference.

Para obter propriedades de trabalho de importação

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta. A guia Conjuntos de imagens é selecionada por padrão.
3. Escolha a guia Importações.

4. Escolha uma tarefa de importação.

   A página Detalhes do trabalho de importação é aberta e exibe propriedades sobre o trabalho de importação.

AWS CLI and SDKs

C++

SDK para 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;
}
```

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS SDK para C++ API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para obter as propriedades de um trabalho de importação dicom

O exemplo de código `get-dicom-import-job` a seguir obtém as propriedades de um trabalho de importação dicom.

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

Saída:

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

Para obter mais informações, consulte Obter propriedades do trabalho de importação no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS CLI Command Reference.

Java

SDK para 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;
   }
```

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS SDK for Java 2.x API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

  return response;
};
```

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS SDK para JavaScript API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte Referência da API Get DICOMImport Job in AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

```
    ENDTRY.
```

- Para obter detalhes da API, consulte a referência da API Get DICOMImport Job in AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Listar trabalhos de importação

Use a `ListDICOMImportJobs` ação para listar trabalhos de importação criados para um armazenamento HealthImaging de dados específico. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `ListDICOMImportJobs`a AWS HealthImaging API Reference.

> **Note**
>
> Os trabalhos de importação são mantidos na lista de trabalhos por 90 dias e depois arquivados.

Para listar as tarefas de importação

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

A página Detalhes do datastore é aberta. A guia Conjuntos de imagens é selecionada por padrão.

3. Escolha a guia Importações para listar todos os trabalhos de importação associados.

AWS CLI and SDKs

CLI

AWS CLI

Para listar trabalhos de importação dicom

O exemplo de código `list-dicom-import-jobs` a seguir lista os trabalhos de importação dicom.

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

Saída:

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

Para obter mais informações, consulte Listar trabalhos de importação no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos na Referência de AWS CLI Comandos.

Java

SDK para 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<>();
   }
```

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos na referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

/**
```

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

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

  const jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page.jobSummaries);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxx: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;
};
```

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos na referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte Referência da API Listar DICOMImport trabalhos no AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        oo_result = lo_mig->listdicomimportjobs( iv_datastoreid =
iv_datastore_id ).
        DATA(lt_jobs) = oo_result->get_jobsummaries( ).
        DATA(lv_count) = lines( lt_jobs ).
        MESSAGE |Found { lv_count } DICOM import jobs.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
```

```
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos no AWS SDK para referência da API SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Acessando conjuntos de imagens com a AWS HealthImaging

O acesso a dados de imagens médicas na AWS HealthImaging normalmente envolve pesquisar um conjunto de imagens com uma chave exclusiva e obter os metadados e quadros de imagem associados (dados em pixels).

> **Importante**
>
> Durante a importação, HealthImaging processa os binários da instância DICOM (`.dcm`arquivos) e os transforma em conjuntos de imagens. Use ações nativas da HealthImaging nuvem (APIs) para gerenciar armazenamentos de dados e conjuntos de imagens. Use HealthImaging a representação dos DICOMweb serviços para retornar DICOMweb respostas.

Os tópicos a seguir explicam como usar ações nativas da HealthImaging nuvem no Console de gerenciamento da AWS, AWS CLI, e AWS SDKs pesquisar conjuntos de imagens e obter suas propriedades, metadados e quadros de imagem associados.

Tópicos

- Noções básicas sobre conjuntos de imagem
- Pesquisar conjuntos de imagens
- Obtendo propriedades do conjunto de imagens
- Obtendo metadados do conjunto de imagens
- Obtendo dados de pixels do conjunto de imagens

Noções básicas sobre conjuntos de imagem

Os conjuntos de imagens AWS se assemelham a uma série DICOM e servem como base para a AWS. HealthImaging Os conjuntos de imagens são criados quando você importa seus dados DICOM para o. HealthImaging O serviço tenta organizar os dados P10 importados de acordo com a hierarquia DICOM de estudo, série e instância.

Os conjuntos de imagem foram introduzidos pelos seguintes motivos:

- Support uma ampla variedade de fluxos de trabalho de imagens médicas (clínicos e não clínicos) por meio da flexibilidade. APIs
- Forneça um mecanismo para armazenar e reconciliar dados duplicados e inconsistentes de forma durável. Os dados P10 importados que estejam em conflito com conjuntos de imagens primárias que já estão em um armazenamento permanecerão como não primários. Depois de resolver os conflitos de metadados, esses dados podem se tornar primários.
- Maximize a segurança do paciente agrupando somente dados relacionados.
- Incentive a limpeza dos dados para ajudar a aumentar a visibilidade das inconsistências. Para obter mais informações, consulte Modificar os conjuntos de imagens.

> **Importante**
>
> O uso clínico dos dados DICOM antes de serem limpos pode resultar em danos ao paciente.

Os menus a seguir descrevem conjuntos de imagens com mais detalhes e fornecem exemplos e diagramas para ajudá-lo a compreender sua funcionalidade e finalidade em. HealthImaging

O que é um conjunto de imagens?

Um conjunto de imagens é um AWS conceito que define um mecanismo de agrupamento abstrato para otimizar dados relacionados de imagens médicas que se assemelham muito a uma série DICOM. Quando você importa seus dados de imagem DICOM P10 para um armazenamento de HealthImaging dados da AWS, eles são transformados em conjuntos de imagens compostos por metadados e quadros de imagem (dados em pixels).

> **Note**
>
> Os metadados do conjunto de imagens são normalizados. Em outras palavras, um conjunto comum de atributos e valores mapeia os elementos de nível Paciente, Estudo e Série listados no Registro de Elementos de Dados DICOM. HealthImaging usa os seguintes elementos DICOM ao agrupar objetos DICOM P10 de entrada em conjuntos de imagens.

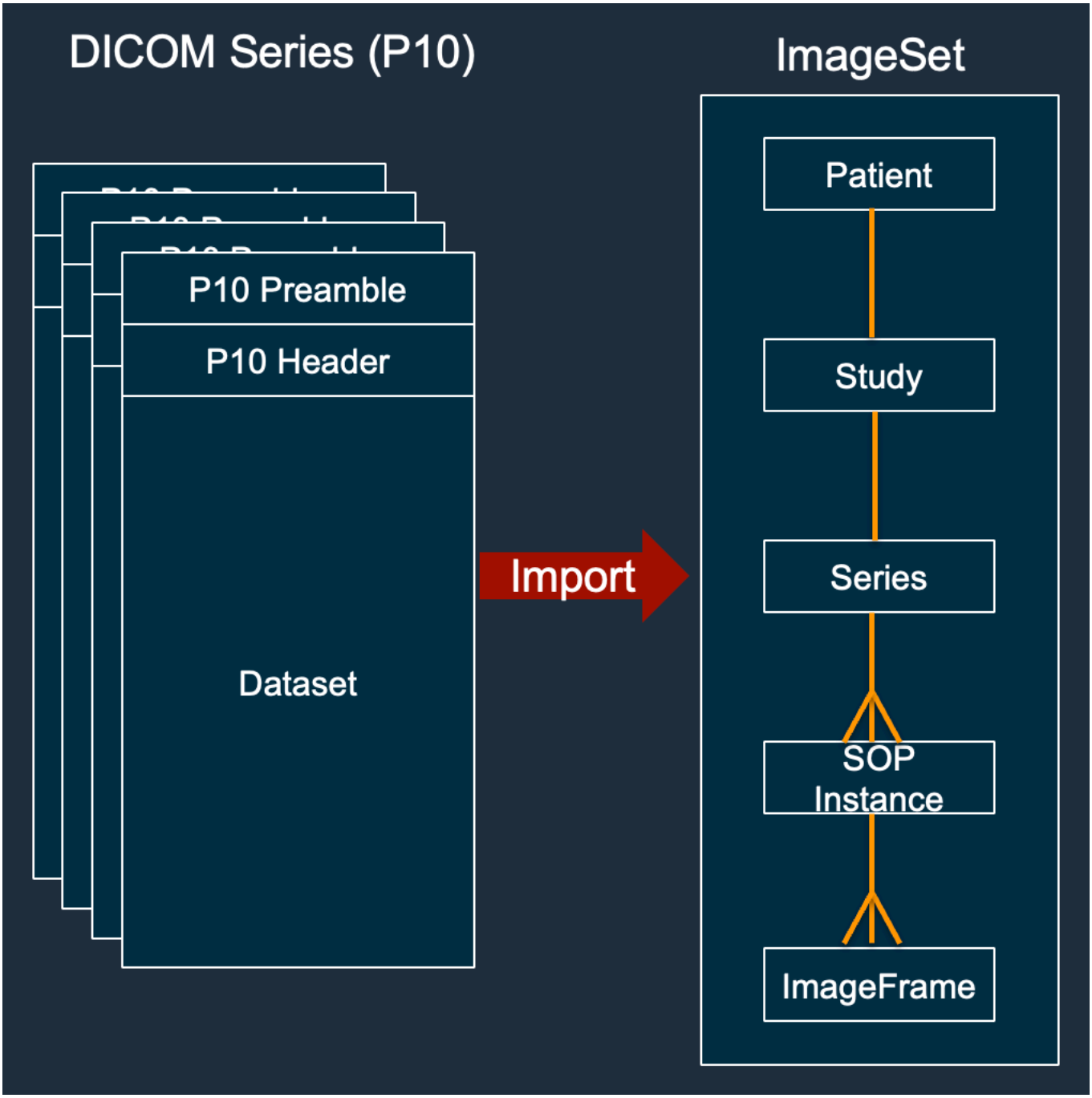
Elementos DICOM usados para criação de conjuntos de imagens

Nome do elemento	Etiqueta do elemento
Elementos do nível de estudo	
`Study Date`	`(0008,0020)`
`Accession Number`	`(0008,0050)`
`Patient ID`	`(0010,0020)`
`Study Instance UID`	`(0020,000D)`
`Study ID`	`(0020,0010)`
Elementos de nível de série	
`Series Instance UID`	`(0020,000E)`
`Series Number`	`(0020,0011)`

Durante a importação, alguns conjuntos de imagens mantêm a codificação da sintaxe de transferência original, enquanto outros são transcodificados para JPEG 2000 (K) de alto rendimento sem perdas por padrão. HTJ2 Se um conjunto de imagens for codificado em HTJ2 K, ele deverá ser decodificado antes da visualização. Para obter mais informações, consulte Sintaxes de transferência compatíveis e Bibliotecas de decodificação de quadros de imagem.
Os quadros de imagem (dados em pixels) são codificados em JPEG 2000 (HTJ2K) de alto rendimento e devem ser decodificados antes da visualização.

Os conjuntos de imagens são AWS recursos, então eles são atribuídos aos Amazon Resource Names (ARNs). Eles podem ser marcados com até 50 pares de valor-chave e podem receber controle de acesso por perfil (RBAC) e controle de acesso por atributo (ABAC) por meio do IAM. Além disso, os conjuntos de imagens são versionados para que todas as alterações sejam preservadas e as versões anteriores possam ser acessadas.

A importação de dados DICOM P10 resulta em conjuntos de imagens que contêm metadados DICOM e quadros de imagem para uma ou mais instâncias de par de objetos de serviço (SOP) na mesma série DICOM.



> **Note**
>
> Trabalhos de importação de DICOM:

- Sempre crie novos conjuntos de imagens ou incremente a versão dos conjuntos de imagens existentes.
- Não desduplique o armazenamento da instância SOP. Cada importação da mesma instância SOP usa armazenamento adicional como um novo conjunto de imagens não primárias ou versão incrementada de um conjunto de imagens primárias existente.
- Organize automaticamente instâncias de SOP com metadados consistentes e não conflitantes como conjuntos de imagens primárias, que contêm instâncias com elementos consistentes de metadados de pacientes, estudos e séries.
  - Se as instâncias que compõem uma série DICOM forem importadas em duas ou mais tarefas de importação e não entrarem em conflito com as instâncias que já estão no armazenamento de dados, todas as instâncias serão organizadas em um conjunto de imagens primárias.
- Crie conjuntos de imagens não primárias contendo dados DICOM P10 que estejam em conflito com conjuntos de imagens primárias que já estão no armazenamento de dados.
- Mantenha os dados recebidos mais recentemente como a versão mais recente de um conjunto de imagens primárias.
  - Se as instâncias que compõem uma série DICOM forem conjuntos de imagens primárias e uma instância for importada novamente, a nova cópia será inserida no conjunto de imagens primário e a versão será incrementada.

Qual é a aparência dos metadados do conjunto de imagens?

Use a `GetImageSetMetadata` ação para recuperar os metadados do conjunto de imagens. Os metadados retornados são compactados com`gzip`, portanto, você deve descompactá-los antes de visualizá-los. Para obter mais informações, consulte Obtendo metadados do conjunto de imagens.

O exemplo a seguir mostra a estrutura dos metadados do conjunto de imagens no formato JSON.

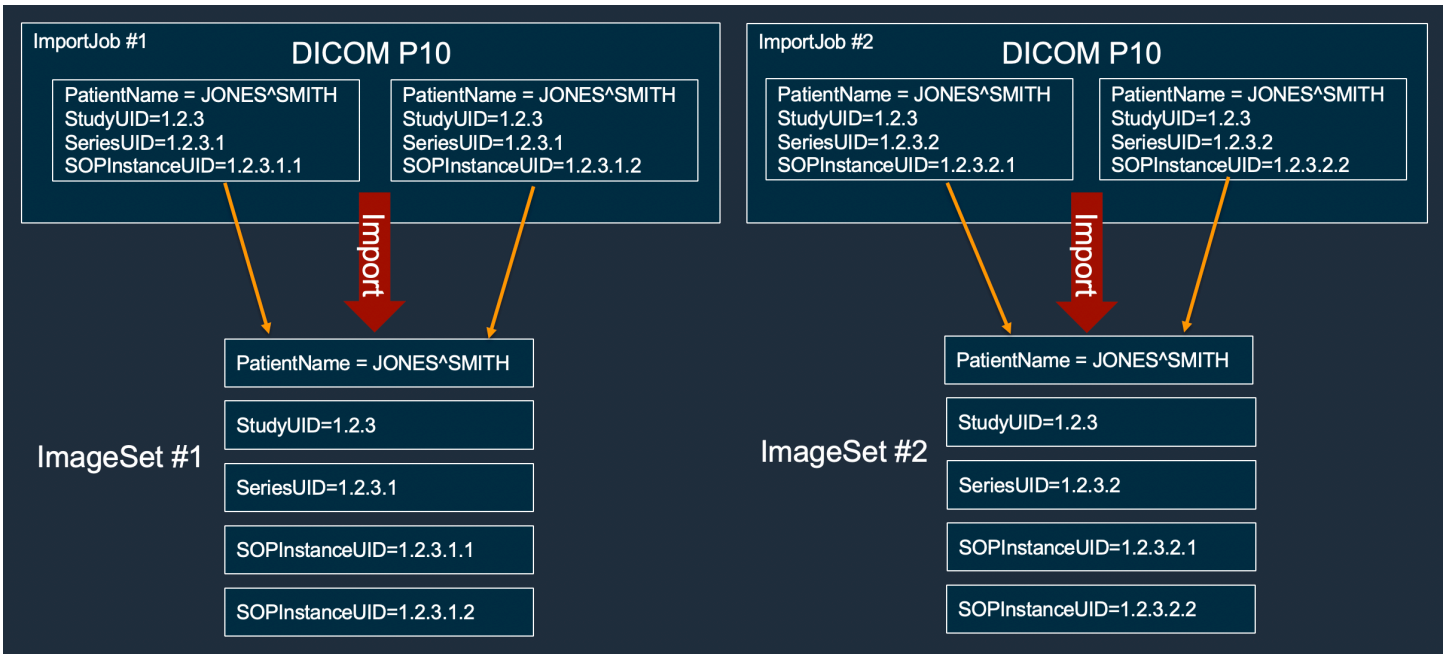
```
{
 "SchemaVersion": "1.1",
 "DatastoreID": "2aa75d103f7f45ab977b0e93f00e6fe9",
 "ImageSetID": "46923b66d5522e4241615ecd64637584",
 "Patient": {
  "DICOM": {
   "PatientBirthDate": null,
   "PatientSex": null,
```

```
   "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
       }
      ]
```

```
      }
     }
    }
   }
  }
 }
}
```

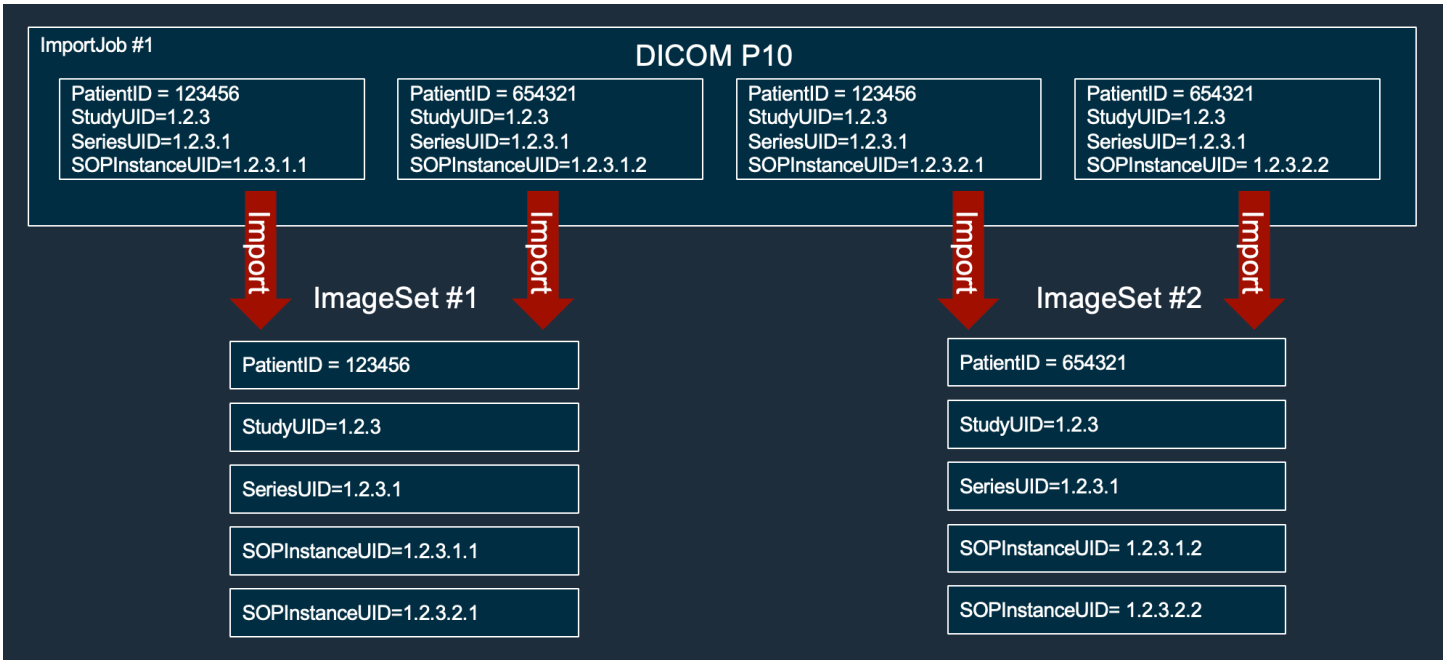
Exemplo de criação de conjunto de imagens: vários trabalhos de importação

O exemplo a seguir mostra como várias tarefas de importação sempre criam novos conjuntos de imagens e nunca são adicionados aos existentes.



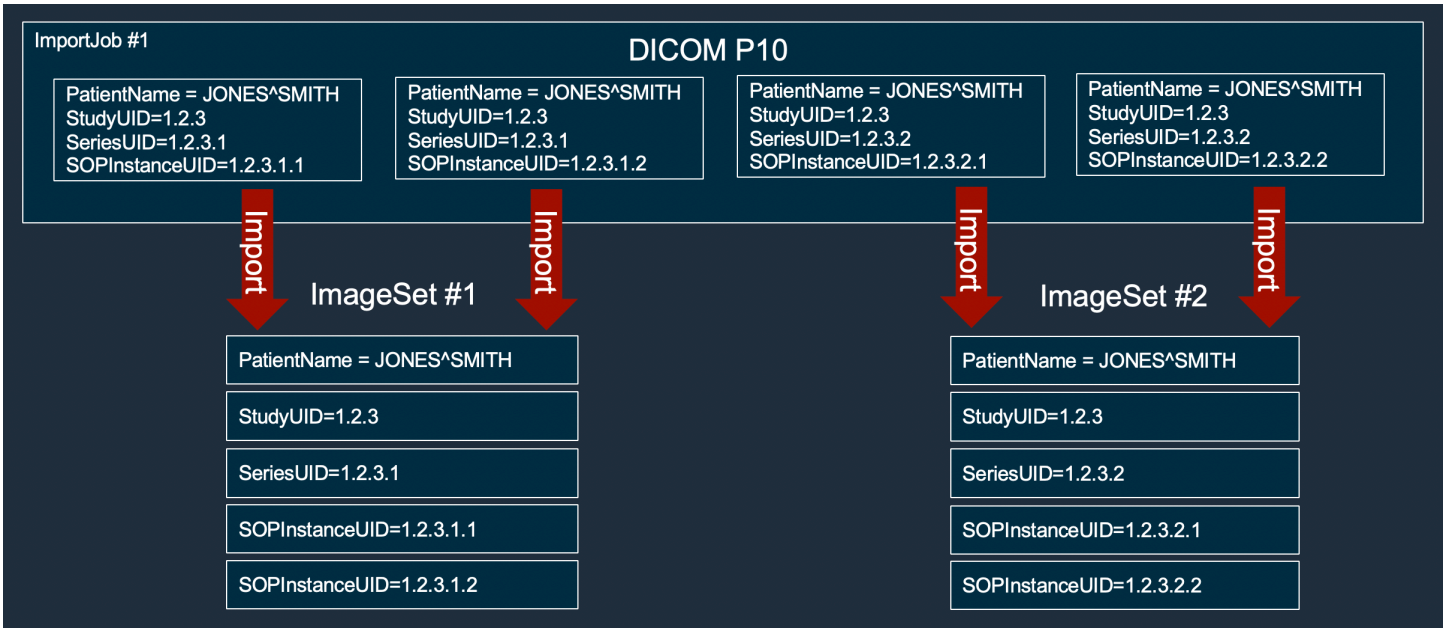
Exemplo de criação de conjunto de imagens: tarefa de importação única com duas variantes

O exemplo a seguir mostra um único trabalho de importação que não seria mesclado em um único conjunto de imagens porque as instâncias 1 e 3 têm um paciente IDs diferente das instâncias 2 e 4. Para resolver isso, você pode usar a `UpdateImageSetMetadata` ação para resolver o conflito de ID do paciente com o conjunto de imagens primárias existente. Depois que os conflitos forem resolvidos, você poderá usar a `CopyImageSet` ação com o argumento `--promoteToPrimary` para adicionar o conjunto de imagens ao conjunto de imagens primárias.



Exemplo de criação de conjunto de imagens: tarefa de importação única com otimização

O exemplo a seguir mostra um único trabalho de importação criando dois conjuntos de imagens para melhorar a produtividade, mesmo que os nomes dos pacientes sejam iguais.



Pesquisar conjuntos de imagens

Use a `SearchImageSets` ação para executar consultas de pesquisa em todos os conjuntos de imagens em um armazenamento de `ACTIVE` HealthImaging dados. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `SearchImageSets`a AWS HealthImaging API Reference.

> **Note**
>
> Lembre-se dos seguintes pontos ao pesquisar conjuntos de imagens.
>
> - `SearchImageSets` aceita um único parâmetro de consulta de pesquisa e retorna uma resposta paginada de todos os conjuntos de imagens que têm os critérios correspondentes. Todas as consultas de intervalo de datas devem ser inseridas como`(lowerBound, upperBound)`.
> - Por padrão, `SearchImageSets` usa o `updatedAt` campo para classificar em ordem decrescente do mais novo para o mais antigo.
> - Se você criou seu armazenamento de dados com uma AWS KMS chave de propriedade do cliente, deverá atualizar sua política de AWS KMS chaves antes de interagir com conjuntos de imagens. Para obter mais informações sobre chaves gerenciadas pelo cliente, consulte Criar uma chave gerenciada pelo cliente.

Para pesquisar conjuntos de imagens

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

> **Note**
>
> Os procedimentos a seguir mostram como pesquisar conjuntos de imagens usando os filtros de `Updated at` propriedades `Series Instance UID` e.

Series Instance UID

Pesquise conjuntos de imagens usando o filtro de **`Series Instance UID`** propriedades

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta e a guia Conjuntos de imagens é selecionada por padrão.
3. Escolha o menu de filtro de propriedades e selecione`Series Instance UID`.
4. No campo Inserir valor para pesquisar, insira (cole) o UID da instância de série de interesse.

   > **Note**
   >
   > Os valores de UID da instância em série devem ser idênticos aos listados no Registro de identificadores exclusivos DICOM (). UIDs Observe que os requisitos incluem uma série de números que contêm pelo menos um ponto entre eles. Períodos não são permitidos no início ou no final da Instância em Série UIDs. Letras e espaços em branco não são permitidos, portanto, tenha cuidado ao copiar e colar. UIDs
5. Escolha o menu Intervalo de datas, selecione um intervalo de datas para o UID da instância de série e escolha Aplicar.
6. Selecione a opção Pesquisar.

   As instâncias em série UIDs que se enquadram no intervalo de datas selecionado são retornadas na ordem mais recente por padrão.

Updated at

Pesquise conjuntos de imagens usando o filtro de **`Updated at`** propriedades

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta e a guia Conjuntos de imagens é selecionada por padrão.
3. Escolha o menu de filtro de propriedades e escolha`Updated at`.

4. Escolha o menu Intervalo de datas, selecione um intervalo de datas do conjunto de imagens e escolha Aplicar.
5. Selecione a opção Pesquisar.

   Os conjuntos de imagens que se enquadram no intervalo de datas selecionado são retornados na ordem mais recente por padrão.

AWS CLI and SDKs

C++

SDK para C++

A função de utilitário para pesquisar conjuntos de imagens.

```
//! 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;
}
```

Caso de uso nº 1: operador EQUAL.

```
        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;
            }
        }
```

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
         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;
            }
        }
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
        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;
            }
        }
```

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

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

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

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

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

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

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

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

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

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

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

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

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

        useCase4SearchCriteria.SetSort(useCase4Sort);

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

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

- Para obter detalhes da API, consulte SearchImageSetsa Referência AWS SDK para C++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: para pesquisar conjuntos de imagens com um operador EQUAL

O exemplo de código `search-image-sets` a seguir usa o operador EQUAL para pesquisar conjuntos de imagens com base em um valor específico.

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

Conteúdo de `search-criteria.json`

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

```
}
```

Saída:

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

Exemplo 2: Para pesquisar conjuntos de imagens com um operador BETWEEN usando DICOMStudy data e DICOMStudy hora

O exemplo de código `search-image-sets` a seguir pesquisa conjuntos de imagens com estudos DICOM gerados entre 1º de janeiro de 1990 (00h) e 1º de janeiro de 2023 (00h).

Observação: DICOMStudy o horário é opcional. Se não estiver presente, 00h (início do dia) é o valor da hora para as datas fornecidas para filtragem.

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

Conteúdo de `search-criteria.json`

```
{
```

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

Saída:

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

Exemplo 3: para pesquisar conjuntos de imagens com um operador BETWEEN usando createdAt (os estudos de tempo persistiam anteriormente)

O exemplo de `search-image-sets` código a seguir pesquisa conjuntos de imagens com estudos DICOM persistentes HealthImaging entre os intervalos de tempo no fuso horário UTC.

Observação: forneça createdAt no formato do exemplo ("1985-04-12T23:20:50.52Z").

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

Conteúdo de `search-criteria.json`

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

Saída:

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

Exemplo 4: Para pesquisar conjuntos de imagens com um operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classificar a resposta em ordem ASC no campo updatedAt

O exemplo de `search-image-sets` código a seguir pesquisa conjuntos de imagens com um operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifica a resposta em ordem ASC no campo updatedAt.

Observação: forneça updatedAt no formato do exemplo ("1985-04-12T23:20:50.52Z").

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

Conteúdo de `search-criteria.json`

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

Saída:

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

Para obter mais informações, consulte Pesquisar conjuntos de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte SearchImageSetsna Referência de AWS CLI Comandos.

Java

SDK para Java 2.x

A função de utilitário para pesquisar conjuntos de imagens.

```
    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;
    }
```

Caso de uso nº 1: operador EQUAL.

```
        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();
        }
```

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
        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();
        }
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
        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();
        }
```

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

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

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

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

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

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

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

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

- Para obter detalhes da API, consulte SearchImageSetsa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

A função de utilitário para pesquisar conjuntos de imagens.

```
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;
};
```

Caso de uso nº 1: operador EQUAL.

```
  const datastoreId = "12345678901234567890123456789012";

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

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

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
  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);
}
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
const datastoreId = "1234567890123456789012345678901 2";

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

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

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

```
const datastoreId = "12345678901234567890123456789012";

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

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

- Para obter detalhes da API, consulte SearchImageSetsa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

A função de utilitário para pesquisar conjuntos de imagens.

```
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
```

Caso de uso nº 1: operador EQUAL.

```
        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}")
```

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
        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}"
        )
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
        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}"
        )
```

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

```
        search_filter = {
            "filters": [
                {
                    "values": [
                        {
                            "updatedAt": datetime.datetime(
```

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

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

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a SearchImageSetsReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->searchimagesets(
      iv_datastoreid = iv_datastore_id
      io_searchcriteria = io_search_criteria ).
    DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).
    DATA(lv_count) = lines( lt_imagesets ).
    MESSAGE |Found { lv_count } image sets.| TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a SearchImageSetsreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Obtendo propriedades do conjunto de imagens

Use a `GetImageSet` ação para retornar as propriedades de uma determinada imagem configurada HealthImaging. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `GetImageSet`a AWS HealthImaging API Reference.

> **Note**
>
> Por padrão, a AWS HealthImaging retorna propriedades para a versão mais recente de um conjunto de imagens. Para visualizar as propriedades de uma versão mais antiga de um conjunto de imagens, forneça o `versionId` com sua solicitação.
> Use `GetDICOMInstance` HealthImaging a representação de um DICOMweb serviço para retornar um binário de instância DICOM (`.dcm`arquivo). Para obter mais informações, consulte Obtendo uma instância DICOM de HealthImaging.

Para obter as propriedades do conjunto de imagens

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta e a guia Conjuntos de imagens é selecionada por padrão.
3. Escolha um conjunto de imagem.

   A página Detalhes do conjunto de imagens é aberta e exibe as propriedades do conjunto de imagens.

AWS CLI and SDKs

CLI

AWS CLI

Para obter as propriedades do conjunto de imagens

O exemplo de código `get-image-set` a seguir obtém as propriedades de um conjunto de imagens.

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

Saída:

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

Para obter mais informações, consulte Como obter propriedades do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetImageSetem Referência de AWS CLI Comandos.

Java

SDK para 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;
    }
```

- Para obter detalhes da API, consulte GetImageSeta Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

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

  return response;
};
```

- Para obter detalhes da API, consulte GetImageSeta Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a GetImageSetReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

```
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a GetImageSetreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Obtendo metadados do conjunto de imagens

Use a `GetImageSetMetadata` ação para recuperar metadados de uma determinada imagem definida em. HealthImaging Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `GetImageSetMetadata`a AWS HealthImaging API Reference.

> **Note**
>
> Por padrão, HealthImaging retorna atributos de metadados para a versão mais recente de um conjunto de imagens. Para visualizar os metadados de uma versão mais antiga de um conjunto de imagens, forneça o `versionId` com sua solicitação.

Os metadados do conjunto de imagens são compactados com `gzip` e retornados como um objeto JSON. Portanto, você deve descompactar o objeto JSON antes de visualizar os metadados normalizados. Para obter mais informações, consulte Normalização de metadados.

Se os metadados de um grande conjunto de imagens ainda estiverem sendo processados após a importação, um 409 `ConflictException` poderá ser retornado. Repita a solicitação após alguns segundos após a conclusão do processamento.

Use `GetDICOMInstanceMetadata` HealthImaging a representação de um DICOMweb serviço para retornar metadados da instância DICOM (`.json`arquivo). Para obter mais informações, consulte Obtendo metadados da instância DICOM de HealthImaging.

Para obter metadados do conjunto de imagens

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta e a guia Conjuntos de imagens é selecionada por padrão.
3. Escolha um conjunto de imagem.

   A página Detalhes do conjunto de imagens é aberta e os metadados do conjunto de imagens são exibidos na seção Visualizador de metadados do conjunto de imagens.

AWS CLI and SDKs

C++

SDK para C++

Função de utilitário para obter metadados do conjunto de imagens.

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

Obter metadados do conjunto de imagens sem versão.

```
        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;
        }
```

Obter metadados do conjunto de imagens com versão.

```
        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;
        }
```

- Para obter detalhes da API, consulte GetImageSetMetadataa Referência AWS SDK para C ++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: para obter os metadados de um conjunto de imagens sem versão

O exemplo de código `get-image-set-metadata` a seguir obtém metadados para um conjunto de imagens sem especificar uma versão.

Observação: `outfile` é um parâmetro obrigatório

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

Os metadados retornados são compactados com o gzip e armazenados no arquivo studymetadata.json.gz. Para visualizar o conteúdo do objeto JSON retornado, você deve primeiro descompactá-lo.

Saída:

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

Exemplo 2: para obter os metadados de um conjunto de imagens com versão

O exemplo de código `get-image-set-metadata` a seguir obtém metadados para um conjunto de imagens com uma versão especificada.

Observação: `outfile` é um parâmetro obrigatório

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

Os metadados retornados são compactados com o gzip e armazenados no arquivo studymetadata.json.gz. Para visualizar o conteúdo do objeto JSON retornado, você deve primeiro descompactá-lo.

Saída:

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

Para obter mais informações, consulte Obter metadados do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetImageSetMetadataem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte GetImageSetMetadataa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

Função de utilitário para obter metadados do conjunto de imagens.

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

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

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

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

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
```

```
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

Obter metadados do conjunto de imagens sem versão.

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

Obter metadados do conjunto de imagens com versão.

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

- Para obter detalhes da API, consulte GetImageSetMetadataa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

Função de utilitário para obter metadados do conjunto de imagens.

```
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
```

Obter metadados do conjunto de imagens sem versão.

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

Obter metadados do conjunto de imagens com versão.

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

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a GetImageSetMetadataReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a GetImageSetMetadatareferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Metadados da sintaxe de transferência

Ao importar dados DICOM, HealthImaging mantém o valor original do atributo de sintaxe de transferência nos metadados do conjunto de imagens. A sintaxe de transferência dos dados DICOM originais importados é armazenada como. `TransferSyntaxUID` HealthImaging usa `StoredTransferSyntaxUID` para indicar o formato usado para codificar dados do quadro de imagem no armazenamento de dados: `1.2.840.10008.1.2.4.202` para armazenamentos de dados habilitados para HTJ2 K (padrão) e para armazenamentos de dados habilitados `1.2.840.10008.1.2.4.90` para JPEG 2000 Lossless.

Obtendo dados de pixels do conjunto de imagens

Um quadro de imagem são os dados de pixel que existem em um conjunto de imagens para formar uma imagem médica 2D. Use a `GetImageFrame` ação para recuperar um quadro de imagem sem perdas JPEG 2000 nativo ou HTJ2 codificado em K para uma determinada imagem configurada. HealthImaging Os menus a seguir fornecem exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `GetImageFrame`a AWS HealthImaging API Reference.

> **Note**
>
> Lembre-se dos seguintes pontos ao usar a `GetImageFrame` ação:

- Durante a importação, HealthImaging retém a codificação de algumas sintaxes de transferência e transcodifica outras para HTJ2 K lossless (padrão) ou JPEG 2000 Lossless. A `GetImageFrame` ação retorna o quadro da imagem na sintaxe de transferência armazenada da instância. Nenhuma transcodificação é realizada durante a recuperação para garantir uma latência mínima de recuperação. Os quadros de imagem talvez precisem ser decodificados antes da visualização em um visualizador de imagens, dependendo da sintaxe de transferência. Para obter mais informações, consulte Sintaxes de transferência compatíveis e Bibliotecas de decodificação de quadros de imagem.
- Para instâncias armazenadas HealthImaging com um ou mais quadros de imagem codificados na família MPEG de sintaxes de transferência (que inclui MPEG-4 AVC/H.264 and HEVC/H .265) MPEG2, a `GetImageFrame` ação retornará um objeto de vídeo na sintaxe de transferência armazenada.
- A sintaxe de transferência dos quadros de imagem é especificada no elemento de resposta do `Content-Type HTTP` cabeçalho. Por exemplo, um quadro de imagem codificado em HTJ2 K teráContent-Type: image/jph header. Para obter mais informações, consulte `GetImageFrame`a AWS HealthImaging API Reference.
- Você também pode usar `GetDICOMInstanceFrames` a representação HealthImaging de um DICOMweb serviço para recuperar quadros de instância DICOM (`multipart`solicitação) para visualizadores e DICOMweb aplicativos compatíveis. Para obter mais informações, consulte Obtendo quadros de instância DICOM de HealthImaging.

Para obter dados de pixels do conjunto de imagens

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

> **Note**
>
> Os quadros de imagem devem ser acessados e decodificados programaticamente, pois não há um visualizador de imagens disponível no Console de gerenciamento da AWS.
> Para obter mais informações sobre decodificação e visualização de quadros de imagem, consulte Bibliotecas de decodificação de quadros de imagem.

AWS CLI and SDKs

C++

SDK para 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();
}
```

- Para obter detalhes da API, consulte GetImageFramea Referência AWS SDK para C++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para obter dados de pixels do conjunto de imagens

O exemplo de código `get-image-frame` a seguir obtém um quadro de imagem.

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

Observação: esse exemplo de código não inclui a saída porque a GetImageFrame ação retorna um fluxo de dados de pixels para o arquivo imageframe.jph. Para obter informações sobre decodificação e visualização de quadros de imagem, consulte Bibliotecas de decodificação HTJ2 K.

Para obter mais informações, consulte Obter dados de pixels do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetImageFramena Referência de AWS CLI Comandos.

Java

SDK para 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);
                }
        }
```

- Para obter detalhes da API, consulte GetImageFramea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte GetImageFramea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a GetImageFrameReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

```
        oo_result = lo_mig->getimageframe(
          iv_datastoreid = iv_datastore_id
          iv_imagesetid = iv_image_set_id
          io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(
            iv_imageframeid = iv_image_frame_id ) ).
        DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).
        MESSAGE 'Image frame retrieved.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image frame not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a GetImageFramereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Modificação de conjuntos de imagens com a AWS HealthImaging

Os trabalhos de importação de DICOM normalmente exigem que você modifique seus conjuntos de imagens pelos seguintes motivos:

- Segurança do paciente
- Consistência de dados
- Reduzir custos de armazenamento

> **Importante**
>
> Durante a importação, HealthImaging processa os binários da instância DICOM (`.dcm`arquivos) e os transforma em conjuntos de imagens. Use ações nativas da HealthImaging nuvem (APIs) para gerenciar armazenamentos de dados e conjuntos de imagens. Use HealthImaging a representação dos DICOMweb serviços para retornar DICOMweb respostas.

HealthImaging fornece vários nativos da nuvem APIs para simplificar o processo de modificação do conjunto de imagens. Os tópicos a seguir descrevem como modificar conjuntos de imagens usando AWS CLI AWS SDKs e.

Tópicos

- Listando versões do conjunto de imagens
- Atualização dos metadados do conjunto de imagens
- Copiar um conjunto de imagens
- Excluir um conjunto de imagens

Listando versões do conjunto de imagens

Use a `ListImageSetVersions` ação para listar o histórico de versões de uma imagem configurada HealthImaging. Os menus a seguir fornecem um procedimento para o Console de gerenciamento

da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `ListImageSetVersions`a AWS HealthImaging API Reference.

> **Note**
>
> A AWS HealthImaging registra todas as alterações feitas em um conjunto de imagens. A atualização dos metadados do conjunto de imagens cria uma nova versão no histórico do conjunto de imagens. Para obter mais informações, consulte Atualização dos metadados do conjunto de imagens.

Para listar as versões de um conjunto de imagens

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta e a guia Conjuntos de imagens é selecionada por padrão.
3. Escolha um conjunto de imagem.

   A página Detalhes do conjunto de imagens é aberta.

   A Versão do conjunto de imagens é exibida na seção Detalhes do conjunto de imagens.

AWS CLI and SDKs

CLI

AWS CLI

Para listar as versões de um conjunto de imagens

O exemplo de código `list-image-set-versions` a seguir lista o histórico de versões de um conjunto de imagens.

```
aws medical-imaging list-image-set-versions \
```

```
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e
```

Saída:


```
{
    "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
        }
    ]
}
```

Para obter mais informações, consulte Listar as versões do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte ListImageSetVersionsem Referência de AWS CLI Comandos.

Java

SDK para 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;
   }
```

- Para obter detalhes da API, consulte ListImageSetVersionsa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte ListImageSetVersionsa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a ListImageSetVersionsReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->listimagesetversions(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
    DATA(lt_versions) = oo_result->get_imagesetpropertieslist( ).
    DATA(lv_count) = lines( lt_versions ).
    MESSAGE |Found { lv_count } image set versions.| TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a ListImageSetVersionsreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

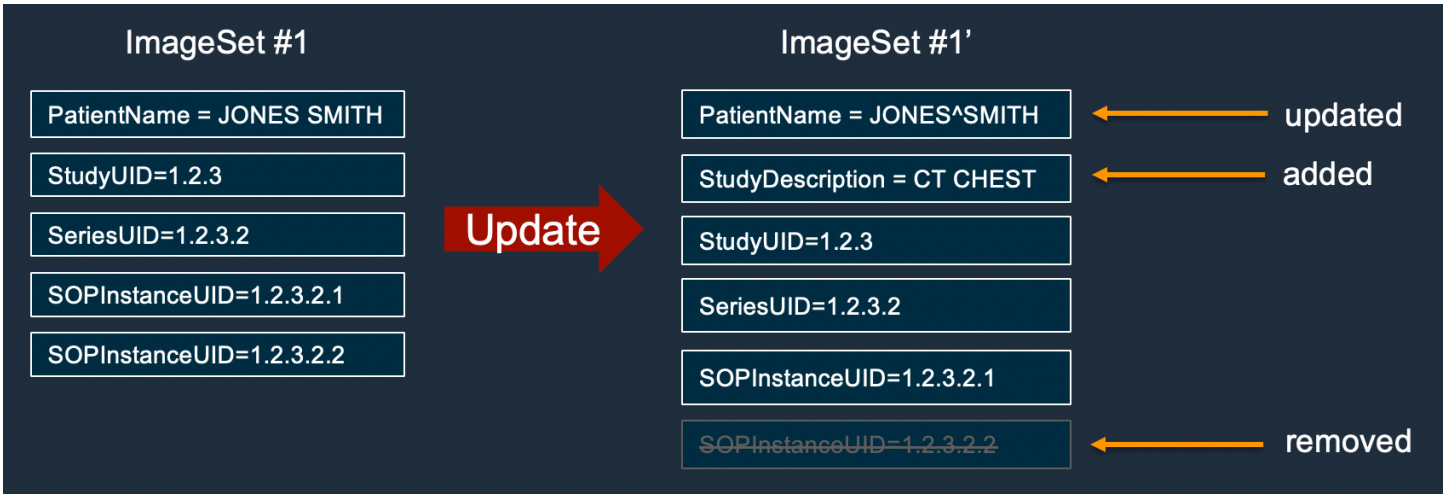
Atualização dos metadados do conjunto de imagens

Use a `UpdateImageSetMetadata` ação para atualizar os metadados do conjunto de imagens na AWS HealthImaging. Você pode usar esse processo assíncrono para adicionar, atualizar e remover atributos de metadados do conjunto de imagens, que são manifestações dos elementos de normalização DICOM criados durante a importação. Usando a ação `UpdateImageSetMetadata`, você também pode remover instâncias de séries e SOP para manter os conjuntos de imagens sincronizados com sistemas externos e para desidentificar os metadados do conjunto de imagens. Para obter mais informações, consulte `UpdateImageSetMetadata`a AWS HealthImaging API Reference.

> **Note**
>
> As importações DICOM do mundo real exigem atualização, adição e remoção de atributos dos metadados do conjunto de imagens. Lembre-se dos seguintes pontos ao atualizar os metadados do conjunto de imagens:
>
> - A atualização dos metadados do conjunto de imagens cria uma nova versão no histórico do conjunto de imagens. Para obter mais informações, consulte Listando versões do conjunto de imagens. Para reverter para um ID de versão anterior do conjunto de imagens, use o `revertToVersionId`parâmetro opcional.
> - A atualização de metadados do conjunto de imagens é um processo assíncrono. Portanto, elementos `imageSetState`de `imageSetWorkflowStatus`resposta estão disponíveis para fornecer o respectivo estado e status de um conjunto de imagens em atualização. Você não pode realizar outras operações de gravação em um conjunto de `LOCKED` imagens.
> - Se a `UpdateImageSetMetadata` ação não for bem-sucedida, ligue e revise o elemento de `message`resposta para ver `common errors.`
> - As restrições do elemento DICOM são aplicadas às atualizações de metadados. O parâmetro de `force`solicitação permite que você atualize elementos de conjuntos de imagens não primários nos casos em que você deseja substituirRestrições de metadados de DICOM.
> - Os elementos de metadados em nível de paciente e série não podem ser atualizados para conjuntos de imagens primárias. Não UpdateImageSet suportará -- `force` para atualizar StudyInstance UID, UID e SeriesInstance SOPInstance UID para conjuntos de imagens primárias.

- Defina o parâmetro de `force`solicitação para forçar a conclusão da `UpdateImageSetMetadata` ação em conjuntos de imagens não primárias. A configuração desse parâmetro permite as seguintes atualizações em um conjunto de imagens:
  - Atualizando `Tag.StudyInstanceUID` os `Tag.StudyID` atributos `Tag.SeriesInstanceUIDTag.SOPInstanceUID`,, e
  - Adicionar, remover ou atualizar elementos de dados DICOM privados no nível da instância
- A ação de promover um conjunto de imagens como primário alterará o ID do conjunto de imagens.

O diagrama a seguir representa os metadados do conjunto de imagens que estão sendo atualizados em HealthImaging.



Para atualizar metadados de um conjunto de imagens

Escolha uma guia com base na sua preferência de acesso à AWS HealthImaging.

AWS CLI and SDKs

CLI

AWS CLI

Exemplo 1: inserir ou atualizar um atributo nos metadados do conjunto de imagens

O exemplo `update-image-set-metadata` a seguir insere ou atualiza um atributo nos metadados do conjunto de imagens.

```
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
```

Conteúdo de `metadata-updates.json`

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

Saída:

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

Exemplo 2: remover um atributo dos metadados do conjunto de imagens

O exemplo `update-image-set-metadata` a seguir remove um atributo dos metadados do conjunto de imagens.

```
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
```

Conteúdo de `metadata-updates.json`

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

Saída:

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

Exemplo 3: remover uma instância dos metadados de um conjunto de imagens

O exemplo `update-image-set-metadata` a seguir remove uma instância dos metadados de um conjunto de imagens.

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

Conteúdo de `metadata-updates.json`

```
{
    "DICOMUpdates": {
```

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

Saída:

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

Exemplo 4: reverter um conjunto de imagens para uma versão anterior

O `update-image-set-metadata` exemplo a seguir mostra como reverter um conjunto de imagens para uma versão anterior. CopyImageSet e UpdateImageSetMetadata ações criam novas versões de conjuntos de imagens.

```
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"}'
```

Saída:

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

```
}
```

Exemplo 5: adicionar um elemento de dados DICOM privado a uma instância

O exemplo `update-image-set-metadata` a seguir mostra como adicionar um elemento privado a uma instância especificada em um conjunto de imagens. O padrão DICOM permite elementos de dados privados para comunicação de informações que não podem estar contidas em elementos de dados padrão. Você pode criar, atualizar e excluir elementos de dados privados com a UpdateImageSetMetadata ação.

```
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
```

Conteúdo de `metadata-updates.json`

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

Saída:

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

Exemplo 6: atualizar um elemento de dados DICOM privado a uma instância

O exemplo `update-image-set-metadata` a seguir mostra como atualizar o valor de um elemento privado que pertence a uma instância em um conjunto de imagens.

```
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
```

Conteúdo de `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":
 {\"00091001\": \"GE_GENESIS_DD\"}}}}}}"
    }
}
```

Saída:

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

Exemplo 7: Para atualizar um SOPInstance UID com o parâmetro force

O `update-image-set-metadata` exemplo a seguir mostra como atualizar um SOPInstance UID usando o parâmetro force para substituir as restrições de metadados DICOM.

```
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
```

Conteúdo de `metadata-updates.json`

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

Saída:

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

Para obter mais informações, consulte Atualização dos metadados do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte UpdateImageSetMetadataem Referência de AWS CLI Comandos.

Java

SDK para 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;
```

```
        }
    }
```

Caso de uso 1: insira ou atualize um atributo.

```
                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);
```

Caso de uso 2: remova um atributo.

```
                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);
```

Caso de uso 3: remova uma instância.

```
                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);
```

Caso de uso 4: reverta para uma versão anterior.

```
                // 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);
```

- Para obter detalhes da API, consulte UpdateImageSetMetadataa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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);
  }
};
```

Caso de uso 1: insira ou atualize um atributo e force a atualização.

```
    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,
  );
```

Caso de uso 2: remova um atributo.

```
  // 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,
);
```

Caso de uso 3: remova uma instância.

```
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,
);
```

Caso de uso 4: reverta para uma versão anterior.

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

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
```

```
      versionID,
      updateMetadata,
    );
```

- Para obter detalhes da API, consulte UpdateImageSetMetadataa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

Caso de uso 1: insira ou atualize um atributo.

```
            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
            )
```

Caso de uso 2: remova um atributo.

```
            # 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
            )
```

Caso de uso 3: remova uma instância.

```
            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
            )
```

Caso de uso 4: reverta para uma versão anterior.

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

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

- Para obter detalhes da API, consulte a UpdateImageSetMetadataReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_latest_version_id = '1'
        " iv_force = abap_false
        oo_result = lo_mig->updateimagesetmetadata(
          iv_datastoreid = iv_datastore_id
          iv_imagesetid = iv_image_set_id
          iv_latestversionid = iv_latest_version_id
          io_updateimagesetmetupdates = io_metadata_updates
          iv_force = iv_force ).
        DATA(lv_new_version) = oo_result->get_latestversionid( ).
        MESSAGE |Image set metadata updated to version: { lv_new_version }.| TYPE
'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
```

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

- Para obter detalhes da API, consulte a UpdateImageSetMetadatareferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Você pode mover instâncias SOP entre conjuntos de imagens, resolver conflitos de elementos de metadados e adicionar ou remover instâncias dos conjuntos de imagens primários usando o `CopyImageSetUpdateImageSetMetadata`, e. `DeleteImageSet` APIs

Você pode remover um conjunto de imagens da coleção primária com a `DeleteImageSet` ação.

Para atualizar os metadados de um conjunto de imagens primário

1. Use a CopyImageSet ação para criar um conjunto de imagens não primárias que seja uma cópia do conjunto de imagens primárias que você deseja modificar. Digamos que isso retorne `103785414bc2c89330f7ce51bbd13f7a` como o ID do conjunto de imagens não primário.

```
        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. Use a UpdateImageSetMetadata ação para fazer alterações no conjunto (`103785414bc2c89330f7ce51bbd13f7a`) de imagens não primárias. Por exemplo, alterar o ID do paciente.

```
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. Exclua o conjunto de imagens primário que você está modificando.

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

4. Use a CopyImageSet ação com o argumento `--promoteToPrimary` para adicionar o conjunto de imagens atualizado à coleção primária.

```
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. Exclua o conjunto de imagens não primárias.

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

Para tornar uma imagem não primária, defina como primária

1. Use a UpdateImageSetMetadata ação para resolver conflitos com conjuntos de imagens primárias existentes.

```
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. Quando os conflitos forem resolvidos, use a CopyImageSet ação com o argumento `--promoteToPrimary` para adicionar o conjunto de imagens à coleção primária do conjunto de imagens.

```
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. Depois de confirmar que a CopyImageSet ação foi bem-sucedida, exclua o conjunto de imagens não primárias de origem.

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

Copiar um conjunto de imagens

Use a `CopyImageSet` ação para copiar um conjunto de imagens HealthImaging. Você usa esse processo assíncrono para copiar o conteúdo de um conjunto de imagens em um conjunto de imagens novo ou existente. Você pode copiar em um novo conjunto de imagens para dividir

um conjunto de imagens, bem como criar uma cópia separada. Você também pode copiar em um conjunto de imagens existente para mesclar dois conjuntos de imagens. Para obter mais informações, consulte `CopyImageSet`a AWS HealthImaging API Reference.

> **Note**
>
> Lembre-se dos seguintes pontos ao usar a `CopyImageSet` ação:
>
> - A `CopyImageSet` ação criará um novo conjunto de imagens ou uma nova versão do`destinationImageSet`. Para obter mais informações, consulte Listando versões do conjunto de imagens.
> - A cópia é um processo assíncrono. Portanto, os elementos de resposta state (`imageSetState`) e status (`imageSetWorkflowStatus`) estão disponíveis para que você saiba qual operação está acontecendo em um conjunto de imagens bloqueado. Outras operações de gravação não podem ser executadas em um conjunto de imagens bloqueado.
> - `CopyImageSet`exige que a instância SOP UIDs seja exclusiva em um conjunto de imagens.
> - Você pode copiar subconjuntos de instâncias SOP usando. `copiableAttributes` Isso permite que você escolha uma ou mais instâncias SOP do `sourceImageSet` para copiar para o. `destinationImageSet`
> - Se a `CopyImageSet` ação não for bem-sucedida, ligue `GetImageSet` e revise a `message`propriedade. Para obter mais informações, consulte Obtendo propriedades do conjunto de imagens.
> - As importações de DICOM do mundo real podem resultar em vários conjuntos de imagens por série DICOM. A `CopyImageSet` ação exige `sourceImageSet` e deve `destinationImageSet` ter metadados consistentes, a menos que o `force`parâmetro opcional seja fornecido.
> - Defina o `force`parâmetro para forçar a operação, mesmo se houver elementos de metadados inconsistentes entre e. `sourceImageSet` `destinationImageSet` Nesses casos, os metadados do paciente, do estudo e da série permanecem inalterados no. `destinationImageSet`

Para copiar um conjunto de imagem

Escolha uma guia com base na sua preferência de acesso à AWS HealthImaging.

AWS CLI and SDKs

CLI

AWS CLI

Exemplo 1: para copiar um conjunto de imagens sem um destino.

O exemplo `copy-image-set` a seguir cria uma cópia duplicada de um conjunto de imagens sem um destino.

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

Saída:

```
{
    "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"
}
```

Exemplo 2: para copiar um conjunto de imagens com um destino.

O exemplo `copy-image-set` a seguir cria uma cópia duplicada de um conjunto de imagens com um destino.

```
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"} }'
```

Saída:

```
{
    "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"
}
```

Exemplo 3: copiar um subconjunto de instâncias de um conjunto de imagens de origem para um conjunto de imagens de destino.

O exemplo `copy-image-set` a seguir copia uma instância DICOM do conjunto de imagens de origem para o conjunto de imagens de destino. O parâmetro force é fornecido para substituir inconsistências nos atributos dos níveis Patient, Study e Series.

```
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
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Copiar um conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte CopyImageSetem Referência de AWS CLI Comandos.

Java

SDK para 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;
        }
    }
```

Função utilitária para criar atributos copiáveis.

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

- Para obter detalhes da API, consulte CopyImageSeta Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

Função de utilitário para copiar um conjunto de imagens.

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

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

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

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

      params.copyImageSetInformation.dicomCopies = copySubsetsJson;
    }

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

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

Copiar um conjunto de imagens sem um destino.

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

Copiar um conjunto de imagens com um destino.

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

Copie um subconjunto de um conjunto de imagens com um destino e force a cópia.

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

- Para obter detalhes da API, consulte CopyImageSeta Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

Função de utilitário para copiar um conjunto de imagens.

```
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"]
```

Copiar um conjunto de imagens sem um destino.

```
            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,
            )
```

Copiar um conjunto de imagens com um destino.

```
            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,
            )
```

Copiar um subconjunto de um conjunto de imagens.

```
            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,
            )
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a CopyImageSetReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_version_id = '1'
        " iv_destination_image_set_id =
'12345678901234567890123456789012345678901234567890' (optional)
        " iv_destination_version_id = '1' (optional)
        " iv_force = abap_false
        DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
          iv_latestversionid = iv_source_version_id ).
        DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(
          io_sourceimageset = lo_source_info ).
        IF iv_destination_image_set_id IS NOT INITIAL AND
           iv_destination_version_id IS NOT INITIAL.
          DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
            iv_imagesetid = iv_destination_image_set_id
            iv_latestversionid = iv_destination_version_id ).
          lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
            io_sourceimageset = lo_source_info
            io_destinationimageset = lo_dest_info ).
        ENDIF.
        oo_result = lo_mig->copyimageset(
          iv_datastoreid = iv_datastore_id
          iv_sourceimagesetid = iv_source_image_set_id
          io_copyimagesetinformation = lo_copy_info
          iv_force = iv_force ).
        DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
        DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
        MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
```

```
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migservicequotaexcdex.
        MESSAGE 'Service quota exceeded.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a CopyImageSetreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Excluir um conjunto de imagens

Use a `DeleteImageSet` ação para excluir uma imagem definida em HealthImaging. Os menus a seguir fornecem um procedimento para o Console de gerenciamento da AWS e exemplos de código para AWS CLI AWS SDKs e. Para obter mais informações, consulte `DeleteImageSet`a AWS HealthImaging API Reference.

Para excluir um conjunto de imagens

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta e a guia Conjuntos de imagens é selecionada por padrão.
3. Escolha um conjunto de imagens e escolha Excluir.

   O modal Excluir conjunto de imagens é aberto.
4. Forneça a ID do conjunto de imagens e escolha Excluir conjunto de imagens.

AWS CLI and SDKs

C++

SDK para 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();
}
```

- Para obter detalhes da API, consulte DeleteImageSeta Referência AWS SDK para C++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para excluir um conjunto de imagens

O exemplo de código `delete-image-set` a seguir exclui um conjunto de imagens.

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

Saída:

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

Para obter mais informações, consulte Excluir um conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte DeleteImageSetem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte DeleteImageSeta Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Para obter detalhes da API, consulte DeleteImageSeta Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

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

- Para obter detalhes da API, consulte a DeleteImageSetReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->deleteimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
    MESSAGE 'Image set deleted.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a DeleteImageSetreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Marcação de recursos com a AWS HealthImaging

Você pode atribuir metadados aos HealthImaging recursos (armazenamentos de dados e conjuntos de imagens) na forma de tags. Cada tag é um rótulo que consiste em um valor e uma chave definida pelo usuário. As tags ajudam a gerenciar, identificar, organizar, pesquisar e filtrar recursos.

> **Importante**
>
> Não armazene informações de saúde protegidas (PHI), informações de identificação pessoal (PII) nem outras informações confidenciais ou sigilosas em tags. As tags não devem ser usadas para dados sensíveis ou privados.

Os tópicos a seguir descrevem como usar operações de HealthImaging marcação usando o Console de gerenciamento da AWS AWS CLI, e. AWS SDKs Para obter mais informações, consulte Como marcar seus AWS recursos no Referência geral da AWS Guia.

Tópicos

- Marcar um recurso
- Listar as tags de um recurso
- Desmarcar um recurso

Marcar um recurso

Use a `TagResource`ação para marcar armazenamentos de dados e conjuntos de imagens na AWS HealthImaging. Os exemplos de código a seguir descrevem como usar a `TagResource` ação com o Console de gerenciamento da AWS AWS CLI, AWS SDKs e. Para obter mais informações, consulte Como marcar seus AWS recursos no Referência geral da AWS Guia.

Como marcar um recurso

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.

2. Escolha um datastore.

   A página Detalhes do datastore é aberta.
3. Escolha a guia Detalhes.
4. Na seção Tags, escolha Gerenciar tags.

   A página Gerenciar tags é aberta.
5. Selecione Adicionar nova tag.
6. Insira uma chave e um valor (opcional).
7. Escolha Salvar alterações.

AWS CLI and SDKs

CLI

AWS CLI

Exemplo 1: para marcar um armazenamento de dados

Os exemplos de código `tag-resource` a seguir marcam um armazenamento de dados.

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

Este comando não produz saída.

Exemplo 2: para marcar um conjunto de imagens

Os exemplos de código `tag-resource` a seguir marcam um conjunto de imagens.

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

Este comando não produz saída.

Para obter mais informações, consulte Como marcar recursos AWS HealthImaging no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte TagResourceem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte TagResourcea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte TagResourcea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a TagResourceReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a TagResourcereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Exemplo de disponibilidade

Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Listar as tags de um recurso

Use a `ListTagsForResource`ação para listar tags para armazenamentos de dados e conjuntos de imagens na AWS HealthImaging. Os exemplos de código a seguir descrevem como usar a `ListTagsForResource` ação com o Console de gerenciamento da AWS AWS CLI, AWS SDKs e. Para obter mais informações, consulte Como marcar seus AWS recursos no Referência geral da AWS Guia.

Para listar as tags para um recurso

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta.
3. Escolha a guia Detalhes.

   Na seção Tags, todas as tags do datastore são listadas.

AWS CLI and SDKs

CLI

AWS CLI

Exemplo 1: para listar as tags de recurso de um armazenamento de dados

O exemplo de código `list-tags-for-resource` a seguir lista as tags de um armazenamento de dados.

```
aws medical-imaging list-tags-for-resource \
```

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

Saída:

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

Exemplo 2: para listar tags de recurso de um conjunto de imagens

O exemplo de código `list-tags-for-resource` a seguir lista as tags de um conjunto de imagens.

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

Saída:

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

Para obter mais informações, consulte Como marcar recursos AWS HealthImaging no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte ListTagsForResourceem Referência de AWS CLI Comandos.

Java

SDK para 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;
    }
```

- Para obter detalhes da API, consulte ListTagsForResourcea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte ListTagsForResourcea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a ListTagsForResourceReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
```

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

- Para obter detalhes da API, consulte a ListTagsForResourcereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Desmarcar um recurso

Use a `UntagResource`ação para desmarcar armazenamentos de dados e conjuntos de imagens na AWS HealthImaging. Os exemplos de código a seguir descrevem como usar a `UntagResource` ação com o Console de gerenciamento da AWS AWS CLI, AWS SDKs e. Para obter mais informações, consulte Como marcar seus AWS recursos no Referência geral da AWS Guia.

Como desmarcar um recurso

Escolha um menu com base na sua preferência de acesso à AWS HealthImaging.

AWS Console

1. Abra a página Armazenamentos de dados do HealthImaging console.
2. Escolha um datastore.

   A página Detalhes do datastore é aberta.
3. Escolha a guia Detalhes.
4. Na seção Tags, escolha Gerenciar tags.

   A página Gerenciar tags é aberta.
5. Escolha Remover ao lado da tag que você quer removida.
6. Escolha Salvar alterações.

AWS CLI and SDKs

CLI

AWS CLI

Exemplo 1: para desmarcar um armazenamento de dados

O exemplo de código `untag-resource` a seguir desmarca um armazenamento de dados.

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

Este comando não produz saída.

Exemplo 2: para desmarcar um conjunto de imagens

O exemplo de código `untag-resource` a seguir desmarca um conjunto de imagens.

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

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

Este comando não produz saída.

Para obter mais informações, consulte Como marcar recursos AWS HealthImaging no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte UntagResourceem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte UntagResourcea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte UntagResourcea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

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

- Para obter detalhes da API, consulte a UntagResourceReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a UntagResourcereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na barra lateral direita desta página.

Exemplos de código para HealthImaging usar AWS SDKs

Os exemplos de código a seguir mostram como usar HealthImaging com um kit AWS de desenvolvimento de software (SDK).

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Exemplos de código

- Exemplos básicos de HealthImaging uso AWS SDKs
  - Olá HealthImaging
  - Ações para HealthImaging usar AWS SDKs
    - Use CopyImageSet com um AWS SDK ou CLI
    - Use CreateDatastore com um AWS SDK ou CLI
    - Use DeleteDatastore com um AWS SDK ou CLI
    - Use DeleteImageSet com um AWS SDK ou CLI
    - Use GetDICOMImportJob com um AWS SDK ou CLI
    - Use GetDatastore com um AWS SDK ou CLI
    - Use GetImageFrame com um AWS SDK ou CLI
    - Use GetImageSet com um AWS SDK ou CLI
    - Use GetImageSetMetadata com um AWS SDK ou CLI
    - Use ListDICOMImportJobs com um AWS SDK ou CLI
    - Use ListDatastores com um AWS SDK ou CLI
    - Use ListImageSetVersions com um AWS SDK ou CLI
    - Use ListTagsForResource com um AWS SDK ou CLI
    - Use SearchImageSets com um AWS SDK ou CLI

    - Use StartDICOMImportJob com um AWS SDK ou CLI
    - Use TagResource com um AWS SDK ou CLI
    - Use UntagResource com um AWS SDK ou CLI
    - Use UpdateImageSetMetadata com um AWS SDK ou CLI
- Cenários para HealthImaging usar AWS SDKs
    - Comece a usar conjuntos de HealthImaging imagens e molduras de imagem usando um AWS SDK
    - Marcar um armazenamento HealthImaging de dados usando um SDK AWS
    - Marcar um conjunto de HealthImaging imagens usando um SDK AWS

Exemplos básicos de HealthImaging uso AWS SDKs

Os exemplos de código a seguir mostram como usar o básico do AWS HealthImaging with AWS SDKs.

Exemplos

- Olá HealthImaging
- Ações para HealthImaging usar AWS SDKs
    - Use CopyImageSet com um AWS SDK ou CLI
    - Use CreateDatastore com um AWS SDK ou CLI
    - Use DeleteDatastore com um AWS SDK ou CLI
    - Use DeleteImageSet com um AWS SDK ou CLI
    - Use GetDICOMImportJob com um AWS SDK ou CLI
    - Use GetDatastore com um AWS SDK ou CLI
    - Use GetImageFrame com um AWS SDK ou CLI
    - Use GetImageSet com um AWS SDK ou CLI
    - Use GetImageSetMetadata com um AWS SDK ou CLI
    - Use ListDICOMImportJobs com um AWS SDK ou CLI
    - Use ListDatastores com um AWS SDK ou CLI
    - Use ListImageSetVersions com um AWS SDK ou CLI
    - Use ListTagsForResource com um AWS SDK ou CLI
    - Use SearchImageSets com um AWS SDK ou CLI

- Use StartDICOMImportJob com um AWS SDK ou CLI
- Use TagResource com um AWS SDK ou CLI
- Use UntagResource com um AWS SDK ou CLI
- Use UpdateImageSetMetadata com um AWS SDK ou CLI

Olá HealthImaging

O exemplo de código a seguir mostra como começar a usar o HealthImaging.

C++

SDK para C++

Código para o CMake arquivo CMake Lists.txt.

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

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

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

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

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

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

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

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

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

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

add_executable(${PROJECT_NAME}
        hello_health_imaging.cpp)

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

Código para o arquivo de origem hello_health_imaging.cpp.

```
#include <aws/core/Aws.h>
#include <aws/medical-imaging/MedicalImagingClient.h>
#include <aws/medical-imaging/model/ListDatastoresRequest.h>

#include <iostream>

/*
 *  A "Hello HealthImaging" starter application which initializes an AWS
 HealthImaging (HealthImaging) client
 *  and lists the HealthImaging data stores in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_health-imaging'
 *
 */
#include <aws/core/auth/AWSCredentialsProviderChain.h>
#include <aws/core/platform/Environment.h>

int main(int argc, char **argv) {
    (void) argc;
    (void) argv;
```

```
    Aws::SDKOptions options;
    //   Optional: change the log level for debugging.
    //   options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;

    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::MedicalImaging::MedicalImagingClient
medicalImagingClient(clientConfig);
        Aws::MedicalImaging::Model::ListDatastoresRequest listDatastoresRequest;

        Aws::Vector<Aws::MedicalImaging::Model::DatastoreSummary>
allDataStoreSummaries;
        Aws::String nextToken; // Used for paginated results.
        do {
            if (!nextToken.empty()) {
                listDatastoresRequest.SetNextToken(nextToken);
            }
            Aws::MedicalImaging::Model::ListDatastoresOutcome
listDatastoresOutcome =
                    medicalImagingClient.ListDatastores(listDatastoresRequest);
            if (listDatastoresOutcome.IsSuccess()) {
                const Aws::Vector<Aws::MedicalImaging::Model::DatastoreSummary>
&dataStoreSummaries =

listDatastoresOutcome.GetResult().GetDatastoreSummaries();
                allDataStoreSummaries.insert(allDataStoreSummaries.cend(),
                                             dataStoreSummaries.cbegin(),
                                             dataStoreSummaries.cend());
                nextToken = listDatastoresOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "ListDatastores error: "
                          << listDatastoresOutcome.GetError().GetMessage() <<
std::endl;
                break;
            }
        } while (!nextToken.empty());

        std::cout << allDataStoreSummaries.size() << " HealthImaging data "
                  << ((allDataStoreSummaries.size() == 1) ?
```

```
                    "store was retrieved." : "stores were retrieved.") <<
 std::endl;

        for (auto const &dataStoreSummary: allDataStoreSummaries) {
            std::cout << "  Datastore: " << dataStoreSummary.GetDatastoreName()
                      << std::endl;
            std::cout << "  Datastore ID: " << dataStoreSummary.GetDatastoreId()
                      << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK para C++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"))
```

- Para obter detalhes da API, consulte a ListDatastoresReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Ações para HealthImaging usar AWS SDKs

Os exemplos de código a seguir demonstram como realizar HealthImaging ações individuais com AWS SDKs. Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções para configurar e executar o código.

Esses trechos chamam a HealthImaging API e são trechos de código de programas maiores que devem ser executados em contexto. É possível ver as ações em contexto em Cenários para HealthImaging usar AWS SDKs.

Os exemplos a seguir incluem apenas as ações mais utilizadas. Para obter uma lista completa, consulte a Referência de APIs do AWS HealthImaging.

Exemplos

- Use CopyImageSet com um AWS SDK ou CLI
- Use CreateDatastore com um AWS SDK ou CLI
- Use DeleteDatastore com um AWS SDK ou CLI
- Use DeleteImageSet com um AWS SDK ou CLI
- Use GetDICOMImportJob com um AWS SDK ou CLI
- Use GetDatastore com um AWS SDK ou CLI
- Use GetImageFrame com um AWS SDK ou CLI
- Use GetImageSet com um AWS SDK ou CLI
- Use GetImageSetMetadata com um AWS SDK ou CLI
- Use ListDICOMImportJobs com um AWS SDK ou CLI
- Use ListDatastores com um AWS SDK ou CLI
- Use ListImageSetVersions com um AWS SDK ou CLI
- Use ListTagsForResource com um AWS SDK ou CLI
- Use SearchImageSets com um AWS SDK ou CLI
- Use StartDICOMImportJob com um AWS SDK ou CLI
- Use TagResource com um AWS SDK ou CLI
- Use UntagResource com um AWS SDK ou CLI
- Use UpdateImageSetMetadata com um AWS SDK ou CLI

Use **`CopyImageSet`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `CopyImageSet`.

CLI

AWS CLI

Exemplo 1: para copiar um conjunto de imagens sem um destino.

O exemplo `copy-image-set` a seguir cria uma cópia duplicada de um conjunto de imagens sem um destino.

```
aws medical-imaging copy-image-set \
```

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

Saída:

```
{
    "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"
}
```

Exemplo 2: para copiar um conjunto de imagens com um destino.

O exemplo `copy-image-set` a seguir cria uma cópia duplicada de um conjunto de imagens com um destino.

```
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"} }'
```

Saída:

```
{
    "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"
}
```

Exemplo 3: copiar um subconjunto de instâncias de um conjunto de imagens de origem para um conjunto de imagens de destino.

O exemplo `copy-image-set` a seguir copia uma instância DICOM do conjunto de imagens de origem para o conjunto de imagens de destino. O parâmetro force é fornecido para substituir inconsistências nos atributos dos níveis Patient, Study e Series.

```
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
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Copiar um conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte CopyImageSetem Referência de AWS CLI Comandos.

Java

SDK para 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;
        }
    }
```

Função utilitária para criar atributos copiáveis.

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

- Para obter detalhes da API, consulte CopyImageSeta Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

Função de utilitário para copiar um conjunto de imagens.

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

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

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

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

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

      params.copyImageSetInformation.dicomCopies = copySubsetsJson;
    }

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

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

Copiar um conjunto de imagens sem um destino.

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

Copiar um conjunto de imagens com um destino.

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

Copie um subconjunto de um conjunto de imagens com um destino e force a cópia.

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

- Para obter detalhes da API, consulte CopyImageSeta Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

Função de utilitário para copiar um conjunto de imagens.

```
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"]
```

Copiar um conjunto de imagens sem um destino.

```
            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,
            )
```

Copiar um conjunto de imagens com um destino.

```
            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,
            )
```

Copiar um subconjunto de um conjunto de imagens.

```
            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,
            )
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a CopyImageSetReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_version_id = '1'
        " iv_destination_image_set_id =
'12345678901234567890123456789012345678901234567890' (optional)
        " iv_destination_version_id = '1' (optional)
        " iv_force = abap_false
        DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
          iv_latestversionid = iv_source_version_id ).
        DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(
          io_sourceimageset = lo_source_info ).
        IF iv_destination_image_set_id IS NOT INITIAL AND
           iv_destination_version_id IS NOT INITIAL.
          DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
            iv_imagesetid = iv_destination_image_set_id
            iv_latestversionid = iv_destination_version_id ).
          lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
            io_sourceimageset = lo_source_info
            io_destinationimageset = lo_dest_info ).
        ENDIF.
        oo_result = lo_mig->copyimageset(
          iv_datastoreid = iv_datastore_id
          iv_sourceimagesetid = iv_source_image_set_id
          io_copyimagesetinformation = lo_copy_info
          iv_force = iv_force ).
        DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
        DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
        MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migservicequotaexcdex.
        MESSAGE 'Service quota exceeded.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
```

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

- Para obter detalhes da API, consulte a CopyImageSetreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`CreateDatastore`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `CreateDatastore`.

Bash

AWS CLI com script Bash

```
###############################################################################
# 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
}
```

- Para obter detalhes da API, consulte CreateDatastoreem Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: criar um datastore

O exemplo de código `create-datastore` a seguir cria um armazenamento de dados com o nome de `my-datastore`. Quando você cria um armazenamento de dados sem especificar um`--lossless-storage-format`, o AWS HealthImaging padrão é HTJ2 K (High Throughput JPEG 2000).

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

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

Saída:

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

Exemplo 2: criar um datastore com o formato de armazenamento JPEG 2000 Lossless

Um datastore configurado com o formato de armazenamento JPEG 2000 Lossless transcodificará e persistirá quadros de imagem sem perdas no formato JPEG 2000. Os quadros de imagem podem então ser recuperados no JPEG 2000 Lossless sem transcodificação. O exemplo de código `create-datastore` a seguir cria um datastore configurado para o formato de armazenamento JPEG 2000 Lossless com o nome `my-datastore`.

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

Saída:

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

Para obter mais informações, consulte Criação de um armazenamento de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte CreateDatastoreem Referência de AWS CLI Comandos.

Java

SDK para 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 "";
   }
```

- Para obter detalhes da API, consulte CreateDatastorea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte CreateDatastorea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a CreateDatastoreReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_name = 'my-datastore-name'
```

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

- Para obter detalhes da API, consulte a CreateDatastorereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`DeleteDatastore`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `DeleteDatastore`.

Bash

AWS CLI com script Bash

```
###############################################################################
# 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
}
```

- Para obter detalhes da API, consulte DeleteDatastoreem Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para excluir um armazenamento de dados

O exemplo de código `delete-datastore` a seguir exclui um armazenamento de dados.

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

Saída:

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

Para obter mais informações, consulte Excluindo um armazenamento de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte DeleteDatastoreem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte DeleteDatastorea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte DeleteDatastorea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a DeleteDatastoreReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->deletedatastore( iv_datastoreid = iv_datastore_id ).
    MESSAGE 'Data store deleted.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict. Data store may contain resources.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Data store not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a DeleteDatastorereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`DeleteImageSet`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `DeleteImageSet`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Começar a usar conjuntos de imagens e quadros de imagem

C++

SDK para 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();
}
```

- Para obter detalhes da API, consulte DeleteImageSeta Referência AWS SDK para C++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para excluir um conjunto de imagens

O exemplo de código `delete-image-set` a seguir exclui um conjunto de imagens.

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

Saída:

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

Para obter mais informações, consulte Excluir um conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte DeleteImageSetem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte DeleteImageSeta Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

/**
```

```
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Para obter detalhes da API, consulte DeleteImageSeta Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a DeleteImageSetReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->deleteimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
    MESSAGE 'Image set deleted.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a DeleteImageSetreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **GetDICOMImportJob** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `GetDICOMImportJob`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Começar a usar conjuntos de imagens e quadros de imagem

C++

SDK para 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;
```

```
}
```

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS SDK para C++ API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para obter as propriedades de um trabalho de importação dicom

O exemplo de código `get-dicom-import-job` a seguir obtém as propriedades de um trabalho de importação dicom.

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

Saída:

```
{
    "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/"
```

```
    }
}
```

Para obter mais informações, consulte Obter propriedades do trabalho de importação no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS CLI Command Reference.

Java

SDK para 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;
   }
```

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS SDK for Java 2.x API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

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

  return response;
};
```

- Para obter detalhes da API, consulte Get DICOMImport Job in AWS SDK para JavaScript API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte Referência da API Get DICOMImport Job in AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_job_id = '12345678901234567890123456789012'
    oo_result = lo_mig->getdicomimportjob(
      iv_datastoreid = iv_datastore_id
      iv_jobid = iv_job_id ).
    DATA(lo_job_props) = oo_result->get_jobproperties( ).
    DATA(lv_job_status) = lo_job_props->get_jobstatus( ).
    MESSAGE |Job status: { lv_job_status }.| TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
```

```
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Job not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a referência da API Get DICOMImport Job in AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`GetDatastore`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `GetDatastore`.

Bash

AWS CLI com script Bash

```
###############################################################################
# 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
}
```

- Para obter detalhes da API, consulte GetDatastoreem Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: obter as propriedades de um datastore

O exemplo de código `get-datastore` a seguir obtém as propriedades de um armazenamento de dados.

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

Saída:

```
{
    "datastoreProperties": {
        "datastoreId": "12345678901234567890123456789012",
        "datastoreName": "TestDatastore123",
        "datastoreStatus": "ACTIVE",
        "losslessStorageFormat": "HTJ2K"
        "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
        "createdAt": "2022-11-15T23:33:09.643000+00:00",
        "updatedAt": "2022-11-15T23:33:09.643000+00:00"
    }
}
```

Exemplo 2: Para configurar as propriedades do armazenamento de dados para JPEG2000

O exemplo de código `get-datastore` a seguir obtém as propriedades de um datastore configurado para o formato de armazenamento JPEG 2000 Lossless.

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

Saída:

```
{
    "datastoreProperties": {
        "datastoreId": "12345678901234567890123456789012",
        "datastoreName": "TestDatastore123",
        "datastoreStatus": "ACTIVE",
        "losslessStorageFormat": "JPEG_2000_LOSSLESS",
        "datastoreArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012",
        "createdAt": "2022-11-15T23:33:09.643000+00:00",
        "updatedAt": "2022-11-15T23:33:09.643000+00:00"
```

```
    }
}
```

Para obter mais informações, consulte Obter propriedades do armazenamento de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetDatastoreem Referência de AWS CLI Comandos.

Java

SDK para 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;
   }
```

- Para obter detalhes da API, consulte GetDatastorea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

- Para obter detalhes da API, consulte GetDatastorea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

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

- Para obter detalhes da API, consulte a GetDatastoreReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->getdatastore( iv_datastoreid = iv_datastore_id ).
    DATA(lo_properties) = oo_result->get_datastoreproperties( ).
    DATA(lv_name) = lo_properties->get_datastorename( ).
    DATA(lv_status) = lo_properties->get_datastorestatus( ).
    MESSAGE 'Data store properties retrieved.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Data store not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a GetDatastorereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`GetImageFrame`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `GetImageFrame`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Começar a usar conjuntos de imagens e quadros de imagem

C++

SDK para 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();
}
```

- Para obter detalhes da API, consulte GetImageFramea Referência AWS SDK para C++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para obter dados de pixels do conjunto de imagens

O exemplo de código `get-image-frame` a seguir obtém um quadro de imagem.

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

Observação: esse exemplo de código não inclui a saída porque a GetImageFrame ação retorna um fluxo de dados de pixels para o arquivo imageframe.jph. Para obter informações sobre decodificação e visualização de quadros de imagem, consulte Bibliotecas de decodificação HTJ2 K.

Para obter mais informações, consulte Obter dados de pixels do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetImageFrameem Referência de AWS CLI Comandos.

Java

SDK para 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);
                        }
                }
```

- Para obter detalhes da API, consulte GetImageFramea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte GetImageFramea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a GetImageFrameReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '1234567890123456789012345678901234567890'
        " iv_image_set_id = '1234567890123456789012345678901234567890'
        " iv_image_frame_id = '1234567890123456789012345678901234567890'
        oo_result = lo_mig->getimageframe(
          iv_datastoreid = iv_datastore_id
          iv_imagesetid = iv_image_set_id
          io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(
            iv_imageframeid = iv_image_frame_id ) ).
        DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).
        MESSAGE 'Image frame retrieved.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image frame not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a GetImageFramereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`GetImageSet`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `GetImageSet`.

CLI

AWS CLI

Para obter as propriedades do conjunto de imagens

O exemplo de código `get-image-set` a seguir obtém as propriedades de um conjunto de imagens.

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

Saída:

```
{
    "versionId": "1",
    "imageSetWorkflowStatus": "COPIED",
    "updatedAt": 1680027253.471,
    "imageSetId": "18f88ac7870584f58d56256646b4d92b",
    "imageSetState": "ACTIVE",
```

```
    "createdAt": 1679592510.753,
    "datastoreId": "12345678901234567890123456789012"
}
```

Para obter mais informações, consulte Como obter propriedades do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetImageSetem Referência de AWS CLI Comandos.

Java

SDK para 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;
   }
```

- Para obter detalhes da API, consulte GetImageSeta Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

```
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Para obter detalhes da API, consulte GetImageSeta Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a GetImageSetReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_version_id = '1' (optional)
        IF iv_version_id IS NOT INITIAL.
          oo_result = lo_mig->getimageset(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id
            iv_versionid = iv_version_id ).
        ELSE.
          oo_result = lo_mig->getimageset(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id ).
        ENDIF.
        DATA(lv_state) = oo_result->get_imagesetstate( ).
        MESSAGE |Image set retrieved with state: { lv_state }.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a GetImageSetreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **GetImageSetMetadata** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `GetImageSetMetadata`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Começar a usar conjuntos de imagens e quadros de imagem

C++

SDK para C++

Função de utilitário para obter metadados do conjunto de imagens.

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

Obter metadados do conjunto de imagens sem versão.

```
        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;
        }
```

Obter metadados do conjunto de imagens com versão.

```
        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;
        }
```

- Para obter detalhes da API, consulte GetImageSetMetadataa Referência AWS SDK para C ++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: para obter os metadados de um conjunto de imagens sem versão

O exemplo de código `get-image-set-metadata` a seguir obtém metadados para um conjunto de imagens sem especificar uma versão.

Observação: `outfile` é um parâmetro obrigatório

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

Os metadados retornados são compactados com o gzip e armazenados no arquivo studymetadata.json.gz. Para visualizar o conteúdo do objeto JSON retornado, você deve primeiro descompactá-lo.

Saída:

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

Exemplo 2: para obter os metadados de um conjunto de imagens com versão

O exemplo de código `get-image-set-metadata` a seguir obtém metadados para um conjunto de imagens com uma versão especificada.

Observação: `outfile` é um parâmetro obrigatório

```
aws medical-imaging get-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
```

```
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --version-id 1 \
    studymetadata.json.gz
```

Os metadados retornados são compactados com o gzip e armazenados no arquivo studymetadata.json.gz. Para visualizar o conteúdo do objeto JSON retornado, você deve primeiro descompactá-lo.

Saída:

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

Para obter mais informações, consulte Obter metadados do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte GetImageSetMetadataem Referência de AWS CLI Comandos.

Java

SDK para 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);
        }
    }
```

- Para obter detalhes da API, consulte GetImageSetMetadataa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

Função de utilitário para obter metadados do conjunto de imagens.

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

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

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

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

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

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

  return response;
};
```

Obter metadados do conjunto de imagens sem versão.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
    );
```

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

Obter metadados do conjunto de imagens com versão.

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

- Para obter detalhes da API, consulte GetImageSetMetadataa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

Função de utilitário para obter metadados do conjunto de imagens.

```
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
```

Obter metadados do conjunto de imagens sem versão.

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

Obter metadados do conjunto de imagens com versão.

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

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a GetImageSetMetadataReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '1234567890123456789012345678901234567890'
        " iv_image_set_id = '1234567890123456789012345678901234567890'
        " iv_version_id = '1' (optional)
        IF iv_version_id IS NOT INITIAL.
```

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

- Para obter detalhes da API, consulte a GetImageSetMetadatareferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`ListDICOMImportJobs`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `ListDICOMImportJobs`.

CLI

AWS CLI

Para listar trabalhos de importação dicom

O exemplo de código `list-dicom-import-jobs` a seguir lista os trabalhos de importação dicom.

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

Saída:

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

Para obter mais informações, consulte Listar trabalhos de importação no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos na Referência de AWS CLI Comandos.

Java

SDK para 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<>();
    }
```

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos na referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

```
    pageSize: 50,
  };

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

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

  return jobSummaries;
};
```

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos na referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte Referência da API Listar DICOMImport trabalhos no AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        oo_result = lo_mig->listdicomimportjobs( iv_datastoreid =
iv_datastore_id ).
        DATA(lt_jobs) = oo_result->get_jobsummaries( ).
        DATA(lv_count) = lines( lt_jobs ).
        MESSAGE |Found { lv_count } DICOM import jobs.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte Listar DICOMImport trabalhos no AWS SDK para referência da API SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`ListDatastores`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `ListDatastores`.

Bash

AWS CLI com script Bash

```
###############################################################################
# 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
}
```

- Para obter detalhes da API, consulte ListDatastoresem Referência de AWS CLI Comandos.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para listar armazenamentos de dados

O exemplo de código `list-datastores` a seguir lista os armazenamentos de dados disponíveis.

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

Saída:

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

Para obter mais informações, consulte Listar armazenamentos de dados no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte ListDatastoresem Referência de AWS CLI Comandos.

Java

SDK para 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;
   }
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

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

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

  return datastoreSummaries;
```

```
};
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a ListDatastoresReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a ListDatastoresreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`ListImageSetVersions`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `ListImageSetVersions`.

CLI

AWS CLI

Para listar as versões de um conjunto de imagens

O exemplo de código `list-image-set-versions` a seguir lista o histórico de versões de um conjunto de imagens.

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

Saída:

```
{
    "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
        }
    ]
}
```

Para obter mais informações, consulte Listar as versões do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte ListImageSetVersionsem Referência de AWS CLI Comandos.

Java

SDK para 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;
    }
```

- Para obter detalhes da API, consulte ListImageSetVersionsa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte ListImageSetVersionsa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a ListImageSetVersionsReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->listimagesetversions(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
    DATA(lt_versions) = oo_result->get_imagesetpropertieslist( ).
    DATA(lv_count) = lines( lt_versions ).
    MESSAGE |Found { lv_count } image set versions.| TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
```

```
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a ListImageSetVersionsreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`ListTagsForResource`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `ListTagsForResource`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Marcar um datastore
- Marcar um conjunto de imagens

CLI

AWS CLI

Exemplo 1: para listar as tags de recurso de um armazenamento de dados

O exemplo de código `list-tags-for-resource` a seguir lista as tags de um armazenamento de dados.

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

Saída:

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

Exemplo 2: para listar tags de recurso de um conjunto de imagens

O exemplo de código `list-tags-for-resource` a seguir lista as tags de um conjunto de imagens.

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

Saída:

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

Para obter mais informações, consulte Como marcar recursos AWS HealthImaging no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte ListTagsForResourceem Referência de AWS CLI Comandos.

Java

SDK para 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;
    }
```

- Para obter detalhes da API, consulte ListTagsForResourcea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte ListTagsForResourcea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a ListTagsForResourceReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

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

- Para obter detalhes da API, consulte a ListTagsForResourcereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`SearchImageSets`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `SearchImageSets`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Começar a usar conjuntos de imagens e quadros de imagem

C++

SDK para C++

A função de utilitário para pesquisar conjuntos de imagens.

```
//! 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;
}
```

Caso de uso nº 1: operador EQUAL.

```
        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;
            }
        }
```

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
         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;
            }
        }
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
       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;
           }
       }
```

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

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

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

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

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

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

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

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

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

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

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

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

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

        useCase4SearchCriteria.SetSort(useCase4Sort);

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

- Para obter detalhes da API, consulte SearchImageSetsa Referência AWS SDK para C++ da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Exemplo 1: para pesquisar conjuntos de imagens com um operador EQUAL

O exemplo de código `search-image-sets` a seguir usa o operador EQUAL para pesquisar conjuntos de imagens com base em um valor específico.

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

Conteúdo de `search-criteria.json`

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

Saída:

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

Exemplo 2: Para pesquisar conjuntos de imagens com um operador BETWEEN usando DICOMStudy data e DICOMStudy hora

O exemplo de código `search-image-sets` a seguir pesquisa conjuntos de imagens com estudos DICOM gerados entre 1º de janeiro de 1990 (00h) e 1º de janeiro de 2023 (00h).

Nota: O DICOMStudy horário é opcional. Se não estiver presente, 00h (início do dia) é o valor da hora para as datas fornecidas para filtragem.

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

Conteúdo de `search-criteria.json`

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

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

Saída:

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

Exemplo 3: para pesquisar conjuntos de imagens com um operador BETWEEN usando createdAt (os estudos de tempo persistiam anteriormente)

O exemplo de `search-image-sets` código a seguir pesquisa conjuntos de imagens com estudos DICOM persistentes HealthImaging entre os intervalos de tempo no fuso horário UTC.

Observação: forneça createdAt no formato do exemplo ("1985-04-12T23:20:50.52Z").

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

Conteúdo de `search-criteria.json`

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

Saída:

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

Exemplo 4: Para pesquisar conjuntos de imagens com um operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classificar a resposta em ordem ASC no campo updatedAt

O exemplo de `search-image-sets` código a seguir pesquisa conjuntos de imagens com um operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifica a resposta em ordem ASC no campo updatedAt.

Observação: forneça updatedAt no formato do exemplo ("1985-04-12T23:20:50.52Z").

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

Conteúdo de `search-criteria.json`

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

Saída:

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

Para obter mais informações, consulte Pesquisar conjuntos de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte SearchImageSetsem Referência de AWS CLI Comandos.

Java

SDK para Java 2.x

A função de utilitário para pesquisar conjuntos de imagens.

```
    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;
    }
```

Caso de uso nº 1: operador EQUAL.

```
        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();
        }
```

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
        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();
        }
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
        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();
        }
```

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

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

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

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

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

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

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

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

- Para obter detalhes da API, consulte SearchImageSetsa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

A função de utilitário para pesquisar conjuntos de imagens.

```
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;
};
```

Caso de uso nº 1: operador EQUAL.

```
const datastoreId = "12345678901234567890123456789012";

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

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

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
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);
}
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
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);
}
```

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

```
const datastoreId = "12345678901234567890123456789012";

try {
```

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

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

- Para obter detalhes da API, consulte SearchImageSetsa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

A função de utilitário para pesquisar conjuntos de imagens.

```
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
```

Caso de uso nº 1: operador EQUAL.

```
        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}")
```

Caso de uso #2: operador BETWEEN usando DICOMStudy data e DICOMStudy hora.

```
        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}"
        )
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
        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}"
        )
```

Caso de uso #4: operador EQUAL em DICOMSeries InstanceUID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

```
        search_filter = {
            "filters": [
                {
                    "values": [
                        {
                            "updatedAt": datetime.datetime(
                                2021, 8, 4, 14, 49, 54, 429000
                            )
                        },
                        {
                            "updatedAt": datetime.datetime.now()
```

```
                            + datetime.timedelta(days=1)
                        },
                    ],
                    "operator": "BETWEEN",
                },
                {
                    "values": [{"DICOMSeriesInstanceUID": series_instance_uid}],
                    "operator": "EQUAL",
                },
            ],
            "sort": {
                "sortOrder": "ASC",
                "sortField": "updatedAt",
            },
        }

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

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a SearchImageSetsReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->searchimagesets(
      iv_datastoreid = iv_datastore_id
      io_searchcriteria = io_search_criteria ).
    DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).
    DATA(lv_count) = lines( lt_imagesets ).
    MESSAGE |Found { lv_count } image sets.| TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obter detalhes da API, consulte a SearchImageSetsreferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **StartDICOMImportJob** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `StartDICOMImportJob`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Começar a usar conjuntos de imagens e quadros de imagem

C++

SDK para 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();
}
```

- Para obter detalhes sobre a API, consulte Start DICOMImport Job in AWS SDK para C++ API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

CLI

AWS CLI

Para iniciar um trabalho de importação dicom

O exemplo de código `start-dicom-import-job` a seguir inicia um trabalho de importação dicom.

```
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"
```

Saída:

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

Para obter mais informações, consulte Iniciando um trabalho de importação no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte Start DICOMImport Job in AWS CLI Command Reference.

Java

SDK para 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 "";
    }
```

- Para obter detalhes sobre a API, consulte Start DICOMImport Job in AWS SDK for Java 2.x API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

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

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

```
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- Para obter detalhes sobre a API, consulte Start DICOMImport Job in AWS SDK para JavaScript API Reference.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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"]
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte Referência da API Start DICOMImport Job in AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_role_arn = 'arn:aws:iam::123456789012:role/ImportJobRole'
    " iv_input_s3_uri = 's3://my-bucket/input/'
    " iv_output_s3_uri = 's3://my-bucket/output/'
    oo_result = lo_mig->startdicomimportjob(
      iv_jobname = iv_job_name
      iv_datastoreid = iv_datastore_id
      iv_dataaccessrolearn = iv_role_arn
      iv_inputs3uri = iv_input_s3_uri
      iv_outputs3uri = iv_output_s3_uri ).
    DATA(lv_job_id) = oo_result->get_jobid( ).
    MESSAGE |DICOM import job started with ID: { lv_job_id }.| TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
  CATCH /aws1/cx_migservicequotaexcdex.
```

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

- Para obter detalhes da API, consulte a referência da API Start DICOMImport Job in AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`TagResource`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `TagResource`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Marcar um datastore
- Marcar um conjunto de imagens

CLI

AWS CLI

Exemplo 1: para marcar um armazenamento de dados

Os exemplos de código `tag-resource` a seguir marcam um armazenamento de dados.

```
aws medical-imaging tag-resource \
```

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

Este comando não produz saída.

Exemplo 2: para marcar um conjunto de imagens

Os exemplos de código `tag-resource` a seguir marcam um conjunto de imagens.

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

Este comando não produz saída.

Para obter mais informações, consulte Como marcar recursos AWS HealthImaging no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte TagResourceem Referência de AWS CLI Comandos.

Java

SDK para 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);
        }
    }
```

- Para obter detalhes da API, consulte TagResourcea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
};
```

- Para obter detalhes da API, consulte TagResourcea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a TagResourceReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a TagResourcereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **`UntagResource`** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `UntagResource`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Marcar um datastore
- Marcar um conjunto de imagens

CLI

AWS CLI

Exemplo 1: para desmarcar um armazenamento de dados

O exemplo de código `untag-resource` a seguir desmarca um armazenamento de dados.

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

Este comando não produz saída.

Exemplo 2: para desmarcar um conjunto de imagens

O exemplo de código `untag-resource` a seguir desmarca um conjunto de imagens.

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

Este comando não produz saída.

Para obter mais informações, consulte Como marcar recursos AWS HealthImaging no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte UntagResourceem Referência de AWS CLI Comandos.

Java

SDK para 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);
       }
   }
```

- Para obter detalhes da API, consulte UntagResourcea Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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;
```

```
};
```

- Para obter detalhes da API, consulte UntagResourcea Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para obter detalhes da API, consulte a UntagResourceReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

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

- Para obter detalhes da API, consulte a UntagResourcereferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Use **UpdateImageSetMetadata** com um AWS SDK ou CLI

Os exemplos de código a seguir mostram como usar o `UpdateImageSetMetadata`.

CLI

AWS CLI

Exemplo 1: inserir ou atualizar um atributo nos metadados do conjunto de imagens

O exemplo `update-image-set-metadata` a seguir insere ou atualiza um atributo nos metadados do conjunto de imagens.

```
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
```

Conteúdo de `metadata-updates.json`

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

Saída:

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

Exemplo 2: remover um atributo dos metadados do conjunto de imagens

O exemplo `update-image-set-metadata` a seguir remove um atributo dos metadados do conjunto de imagens.

```
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
```

Conteúdo de `metadata-updates.json`

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

Saída:

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

```
}
```

Exemplo 3: remover uma instância dos metadados de um conjunto de imagens

O exemplo `update-image-set-metadata` a seguir remove uma instância dos metadados de um conjunto de imagens.

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

Conteúdo de `metadata-updates.json`

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

Saída:

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

Exemplo 4: reverter um conjunto de imagens para uma versão anterior

O `update-image-set-metadata` exemplo a seguir mostra como reverter um conjunto de imagens para uma versão anterior. CopyImageSet e UpdateImageSetMetadata as ações criam novas versões dos conjuntos de imagens.

```
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"}'
```

Saída:

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

Exemplo 5: adicionar um elemento de dados DICOM privado a uma instância

O exemplo `update-image-set-metadata` a seguir mostra como adicionar um elemento privado a uma instância especificada em um conjunto de imagens. O padrão DICOM permite elementos de dados privados para comunicação de informações que não podem estar contidas em elementos de dados padrão. Você pode criar, atualizar e excluir elementos de dados privados com a UpdateImageSetMetadata ação.

```
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
```

Conteúdo de `metadata-updates.json`

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

```
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":
 {\"001910F9\": \"97\"},\"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}"
    }
}
```

Saída:

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

Exemplo 6: atualizar um elemento de dados DICOM privado a uma instância

O exemplo `update-image-set-metadata` a seguir mostra como atualizar o valor de um elemento privado que pertence a uma instância em um conjunto de imagens.

```
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
```

Conteúdo de `metadata-updates.json`

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances
\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":
 {\"00091001\": \"GE_GENESIS_DD\"}}}}}}"
    }
}
```

Saída:

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

Exemplo 7: Para atualizar um SOPInstance UID com o parâmetro force

O `update-image-set-metadata` exemplo a seguir mostra como atualizar um SOPInstance UID usando o parâmetro force para substituir as restrições de metadados DICOM.

```
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
```

Conteúdo de `metadata-updates.json`

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

Saída:

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
```

```
    "updatedAt": 1680042257.908,
    "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

Para obter mais informações, consulte Atualização dos metadados do conjunto de imagens no Guia do AWS HealthImaging desenvolvedor.

- Para obter detalhes da API, consulte UpdateImageSetMetadataem Referência de AWS CLI Comandos.

Java

SDK para 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;
        }
    }
```

Caso de uso 1: insira ou atualize um atributo.

```
                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);
```

Caso de uso 2: remova um atributo.

```
                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);
```

Caso de uso 3: remova uma instância.

```
                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);
```

Caso de uso 4: reverta para uma versão anterior.

```
                // 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);
```

- Para obter detalhes da API, consulte UpdateImageSetMetadataa Referência AWS SDK for Java 2.x da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para 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);
  }
};
```

Caso de uso 1: insira ou atualize um atributo e force a atualização.

```
    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,
    );
```

Caso de uso 2: remova um atributo.

```
    // 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,
);
```

Caso de uso 3: remova uma instância.

```
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,
```

```
    );
```

Caso de uso 4: reverta para uma versão anterior.

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

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

- Para obter detalhes da API, consulte UpdateImageSetMetadataa Referência AWS SDK para JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para 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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

Caso de uso 1: insira ou atualize um atributo.

```
            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
        )
```

Caso de uso 2: remova um atributo.

```
        # 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
        )
```

Caso de uso 3: remova uma instância.

```
        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
        )
```

Caso de uso 4: reverta para uma versão anterior.

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

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

- Para obter detalhes da API, consulte a UpdateImageSetMetadataReferência da API AWS SDK for Python (Boto3).

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    " iv_latest_version_id = '1'
    " iv_force = abap_false
    oo_result = lo_mig->updateimagesetmetadata(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id
      iv_latestversionid = iv_latest_version_id
      io_updateimagesetmetupdates = io_metadata_updates
```

```
          iv_force = iv_force ).
        DATA(lv_new_version) = oo_result->get_latestversionid( ).
        MESSAGE |Image set metadata updated to version: { lv_new_version }.| TYPE
'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migservicequotaexcdex.
        MESSAGE 'Service quota exceeded.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- Para obter detalhes da API, consulte a UpdateImageSetMetadatareferência da API AWS SDK for SAP ABAP.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Cenários para HealthImaging usar AWS SDKs

Os exemplos de código a seguir mostram como implementar cenários comuns em HealthImaging with AWS SDKs. Esses cenários mostram como realizar tarefas específicas chamando várias funções internas HealthImaging ou combinadas com outras Serviços da AWS. Cada cenário inclui um link para o código-fonte completo, onde podem ser encontradas instruções sobre como configurar e executar o código.

Os cenários têm como alvo um nível intermediário de experiência para ajudar você a compreender ações de serviço em contexto.

Exemplos

- Comece a usar conjuntos de HealthImaging imagens e molduras de imagem usando um AWS SDK
- Marcar um armazenamento HealthImaging de dados usando um SDK AWS
- Marcar um conjunto de HealthImaging imagens usando um SDK AWS

Comece a usar conjuntos de HealthImaging imagens e molduras de imagem usando um AWS SDK

Os exemplos de código a seguir mostram como importar arquivos DICOM e baixar molduras de imagem em HealthImaging.

A implementação é estruturada como uma aplicação da linha de comando.

- Configure recursos para uma importação DICOM.
- Importe arquivos DICOM para um armazenamento de dados.
- Recupere o conjunto de imagens IDs para o trabalho de importação.
- Recupere a moldura da imagem IDs para os conjuntos de imagens.
- Baixe, decodifique e verifique os quadros de imagem.
- Limpe recursos.

C++

SDK para C++

Crie uma CloudFormation pilha com os recursos necessários.

```
    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: ");
   }
```

Copie arquivos DICOM para o bucket de importação do Amazon S3.

```
   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;
    }
```

Importe os arquivos DICOM para o armazenamento de dados do Amazon S3.

```
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;
}
```

Obtenha conjuntos de imagens criados pelo trabalho de importação DICOM.

```
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;
}
```

Obtenha informações sobre os quadros de imagem de conjuntos de imagens.

```
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();
}
```

Baixe, decodifique e verifique os quadros de imagem.

```
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;
}
```

Limpe recursos.

```
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;
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para C++ .
  - DeleteImageSet
  - Consiga DICOMImport um emprego
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Start DICOMImport Job

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

Orquestre etapas (index.js).

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
```

```
import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
```

```
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
```

```
    name: "Health Imaging Workflow",
    description:
      "Work with DICOM images using an AWS Health Imaging data store.",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes]
 [-v|--verbose]",
  });
}
```

Implante recursos (deploy-steps.js).

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
```

```
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
```

```
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
```

```
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

Copie arquivos DICOM (dataset-steps.js).

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
```

```
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = "input/";
```

```
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

Inicie a importação para o datastore (import-steps.js).

```
import {
  MedicalImagingClient,
```

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

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });
```

```
    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

Obtenha o conjunto de imagens IDs (image-set-steps.js-).

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
```

```
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce((ids, next) => {
        return Object.assign({}, ids, {
          [next.imageSetId]: next.imageSetId,
```

```
        });
      }, {});
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

Obtenha a moldura da imagem IDs (image-frame-steps.js).

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node:util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
```

```
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
```

```
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );
```

```
      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
);
```

Verifique os quadros de imagens (verify-steps.js). A biblioteca AWS HealthImaging Pixel Data Verification foi usada para verificação.

```
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}`,
                  ),
```

```
                );
              }
            });
          });
        }
      }
    }
  },
);
```

Destrua recursos (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,
  },
);
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript .
  - DeleteImageSet
  - Consiga DICOMImport um emprego
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Start DICOMImport Job

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

Crie uma CloudFormation pilha com os recursos necessários.

```
    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
```

Copie arquivos DICOM para o bucket de importação do Amazon S3.

```
   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.")
```

Importe os arquivos DICOM para o armazenamento de dados do Amazon S3.

```
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"]
```

Obtenha conjuntos de imagens criados pelo trabalho de importação DICOM.

```
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
```

Obtenha informações sobre os quadros de imagem de conjuntos de imagens.

```
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
```

Baixe, decodifique e verifique os quadros de imagem.

```
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
```

Limpe recursos.

```
    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
```

- Para ver detalhes da API, consulte os tópicos a seguir na Referência da API do SDK da AWS para Python (Boto3).
  - DeleteImageSet
  - Consiga DICOMImport um emprego
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Start DICOMImport Job

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Marcar um armazenamento HealthImaging de dados usando um SDK AWS

Os exemplos de código a seguir mostram como marcar um armazenamento HealthImaging de dados.

Java

SDK para Java 2.x

Marcar um datastore.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

A função de utilitário para marcar um recurso.

```
   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);
        }
    }
```

Listar tags para um datastore.

```
                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());
                }
```

A função de utilitário para listar as tags de um recurso.

```
   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;
   }
```

Desmarcar um datastore.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
```

```
                                Collections.singletonList("Deployment"));
```

A função de utilitário para desmarcar um recurso.

```
   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);
       }
   }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

Marcar um datastore.

```
  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);
  }
```

A função de utilitário para marcar um recurso.

```
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;
};
```

Listar tags para um datastore.

```
  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);
  }
```

A função de utilitário para listar as tags de um recurso.

```
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;
};
```

Desmarcar um datastore.

```
  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);
  }
```

A função de utilitário para desmarcar um recurso.

```
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;
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

Marcar um datastore.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

A função de utilitário para marcar um recurso.

```
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
```

Listar tags para um datastore.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

A função de utilitário para listar as tags de um recurso.

```
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"]
```

Desmarcar um datastore.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

A função de utilitário para desmarcar um recurso.

```
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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para ver detalhes da API, consulte os tópicos a seguir na Referência da API do SDK da AWS para Python (Boto3).
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Marcar um conjunto de HealthImaging imagens usando um SDK AWS

Os exemplos de código a seguir mostram como marcar um conjunto de HealthImaging imagens.

Java

SDK para Java 2.x

Marcar um conjunto de imagens

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

A função de utilitário para marcar um recurso.

```
   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);
        }
    }
```

Listar tags para um conjunto de imagens

```
                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());
                }
```

A função de utilitário para listar as tags de um recurso.

```
   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;
   }
```

Desmarcar um conjunto de imagens

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
```

```
                UntagResource.untagMedicalImagingResource(medicalImagingClient,
imageSetArn,
                                Collections.singletonList("Deployment"));
```

A função de utilitário para desmarcar um recurso.

```
   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);
       }
   }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

JavaScript

SDK para JavaScript (v3)

Marcar um conjunto de imagens

```
  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);
  }
```

A função de utilitário para marcar um recurso.

```
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;
};
```

Listar tags para um conjunto de imagens

```
  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);
  }
```

A função de utilitário para listar as tags de um recurso.

```
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;
};
```

Desmarcar um conjunto de imagens

```
  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);
  }
```

A função de utilitário para desmarcar um recurso.

```
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;
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Python

SDK para Python (Boto3)

Marcar um conjunto de imagens

```
    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"})
```

A função de utilitário para marcar um recurso.

```
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
```

Listar tags para um conjunto de imagens

```
    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)
```

A função de utilitário para listar as tags de um recurso.

```
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"]
```

Desmarcar um conjunto de imagens

```
    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"])
```

A função de utilitário para desmarcar um recurso.

```
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
```

O código a seguir instancia o MedicalImagingWrapper objeto.

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

- Para ver detalhes da API, consulte os tópicos a seguir na Referência da API do SDK da AWS para Python (Boto3).
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Para obter uma lista completa dos guias do desenvolvedor do AWS SDK e exemplos de código, consulteUsando esse serviço com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Monitoramento da AWS HealthImaging

O monitoramento e o registro em log são partes importantes da manutenção da segurança, confiabilidade, disponibilidade e desempenho da AWS HealthImaging. AWS fornece as seguintes ferramentas de registro e monitoramento para observar HealthImaging, relatar quando algo está errado e realizar ações automáticas quando apropriado:

- AWS CloudTrailcaptura chamadas de API e eventos relacionados feitos por ou em nome de sua AWS conta e entrega os arquivos de log para um bucket do Amazon S3 que você especificar. Você pode identificar quais usuários e contas ligaram AWS, o endereço IP de origem a partir do qual as chamadas foram feitas e quando elas ocorreram. Para obter mais informações, consulte o Guia do usuário do AWS CloudTrail.
- A Amazon CloudWatch monitora seus AWS recursos e os aplicativos em que você executa AWS em tempo real. Você pode coletar e rastrear métricas, criar painéis personalizados e definir alarmes que o notificam ou que realizam ações quando uma métrica especificada atinge um limite definido. Por exemplo, você pode CloudWatch rastrear o uso da CPU ou outras métricas de suas EC2 instâncias da Amazon e iniciar automaticamente novas instâncias quando necessário. Para obter mais informações, consulte o Guia CloudWatch do usuário da Amazon.
- EventBridgeA Amazon é um serviço de ônibus de eventos sem servidor que facilita a conexão de seus aplicativos com dados de várias fontes. EventBridge fornece um fluxo de dados em tempo real de seus próprios aplicativos, aplicativos Software-as-a-Service (SaaS) e AWS serviços e encaminha esses dados para destinos como o Lambda. Isso permite monitorar eventos que ocorram em serviços e criem arquiteturas orientadas a eventos. Para obter mais informações, consulte o Guia EventBridge do usuário da Amazon.

Tópicos

- Usando AWS CloudTrail com HealthImaging
- Usando a Amazon CloudWatch com HealthImaging
- Usando a Amazon EventBridge com HealthImaging

Usando AWS CloudTrail com HealthImaging

HealthImaging A AWS está integrada com AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou AWS serviço em HealthImaging. CloudTrail

captura todas as chamadas de API HealthImaging como eventos. As chamadas capturadas incluem chamadas do HealthImaging console e chamadas de código para as operações HealthImaging da API. Se você criar uma trilha, poderá ativar a entrega contínua de CloudTrail eventos para um bucket do Amazon S3, incluindo eventos para. HealthImaging Se você não configurar uma trilha, ainda poderá ver os eventos mais recentes no CloudTrail console no Histórico de eventos. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita HealthImaging, o endereço IP do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais sobre isso CloudTrail, consulte o Guia AWS CloudTrail do usuário.

Criar uma trilha

CloudTrail é ativado para você Conta da AWS quando você cria a conta. Quando a atividade ocorre em HealthImaging, essa atividade é registrada em um CloudTrail evento junto com outros eventos AWS de serviço no histórico de eventos. Você pode exibir, pesquisar e baixar eventos recentes em sua Conta da AWS. Para obter mais informações, consulte Visualização de eventos com histórico de CloudTrail eventos.

> **Note**
>
> Para visualizar o histórico de CloudTrail eventos da AWS HealthImaging no Console de gerenciamento da AWS, acesse o menu Pesquisar atributos, selecione Origem do evento e escolha`medical-imaging.amazonaws.com`.

Para um registro contínuo dos eventos em sua Conta da AWS, incluindo eventos para HealthImaging, crie uma trilha. Uma trilha permite CloudTrail entregar arquivos de log para um bucket do Amazon S3. Por padrão, quando você cria uma trilha no console, ela é aplicada a todas as Regiões da AWS. A trilha registra eventos de todas as regiões na AWS partição e entrega os arquivos de log ao bucket do Amazon S3 que você especificar. Além disso, você pode configurar outros AWS serviços para analisar e agir com base nos dados de eventos coletados nos CloudTrail registros. Para saber mais, consulte:

- Visão geral da criação de uma trilha
- Serviços e integrações compatíveis com o CloudTrail
- Configurando notificações do Amazon SNS para CloudTrail

- Recebendo arquivos de CloudTrail log de várias regiões e Recebendo arquivos de CloudTrail log de várias contas

> **Note**
>
> A AWS HealthImaging oferece suporte a dois tipos de CloudTrail eventos: eventos de gerenciamento e eventos de dados. Eventos de gerenciamento são os eventos gerais que todo AWS serviço gera, inclusive HealthImaging. Por padrão, o registro é aplicado aos eventos de gerenciamento de cada chamada de HealthImaging API que o tenha ativado. Os eventos de dados são faturáveis e geralmente reservados para aqueles APIs que têm altas transações por segundo (tps), portanto, você pode optar por não ter CloudTrail registros para fins de custo.
>
> Com HealthImaging, todas as ações de API nativas listadas na Referência de HealthImaging API da AWS são classificadas como eventos de gerenciamento, com exceção de `GetImageFrame`. A `GetImageFrame` ação é integrada CloudTrail como um evento de dados e, portanto, deve ser ativada. Para obter mais informações, consulte Registrar eventos de dados, no Guia do usuário do AWS CloudTrail .
>
> DICOMweb As ações da API WADO-RS são classificadas como eventos de dados em CloudTrail, portanto, você deve aceitá-las. Para obter mais informações, consulte Recuperando dados DICOM de HealthImaging Registrar eventos de dados no Guia AWS CloudTrail do usuário.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar o seguinte:

- Se a solicitação foi feita com credenciais de usuário root ou AWS Identity and Access Management (IAM).
- Se a solicitação foi feita com credenciais de segurança temporárias de um perfil ou de um usuário federado.
- Se a solicitação foi feita por outro AWS serviço.

Para obter mais informações, consulte o CloudTrail`userIdentity`elemento.

Noções básicas sobre entradas de log

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log para um bucket do Amazon S3 que você especificar. CloudTrail os arquivos de log contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte e inclui informações sobre a ação solicitada, a data e a hora da ação, os parâmetros da solicitação e assim por diante. CloudTrail os arquivos de log não são um rastreamento de pilha ordenado das chamadas públicas de API, portanto, eles não aparecem em nenhuma ordem específica.

O exemplo a seguir mostra uma entrada de CloudTrail registro HealthImaging que demonstra a `GetDICOMImportJob` ação.

```
{
    "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"
}
```


Usando a Amazon CloudWatch com HealthImaging

Você pode monitorar a AWS HealthImaging usando CloudWatch, que coleta dados brutos e os processa em métricas legíveis, quase em tempo real. Essas estatísticas são mantidas por 15 meses, de maneira que você possa usar informações históricas e ter uma perspectiva melhor de como sua aplicação web ou o serviço está se saindo. Você também pode definir alarmes que observam determinados limites e enviam notificações ou realizam ações quando esses limites são atingidos. Para obter mais informações, consulte o Guia CloudWatch do usuário da Amazon.

HealthImaging publica os seguintes tipos de métricas para CloudWatch, no `AWS/HealthImaging` namespace:

- Métricas de API - Contagem de chamadas para operações de HealthImaging API
- HealthImaging métricas - Armazenamento de dados e uso de recursos em nível de conta

> **Note**
>
> - As métricas são relatadas para a maioria HealthImaging APIs.
> - HealthImaging as métricas só estão disponíveis para armazenamentos de dados criados após 9 de fevereiro de 2026 ou por meio da apresentação de um caso de suporte.

Métricas da HealthImaging API da AWS

As tabelas a seguir listam as métricas e dimensões do HealthImaging. Cada uma é apresentada como uma contagem de frequência para um intervalo de datas especificado pelo usuário.

Métrica

Metrics	Description
`CallCount`	O número de chamadas para APIs. Isso pode ser relatado para a conta ou para um datastore especificado. Unidades: contagem Estatísticas válidas: Sum, Count Dimensões: operação, ID do datastore, tipo de datastore

Você pode obter métricas HealthImaging com a Console de gerenciamento da AWS AWS CLI, a ou a CloudWatch API. Você pode usar a CloudWatch API por meio de um dos kits de desenvolvimento de software da Amazon AWS (SDKs) ou das ferramentas de CloudWatch API. O HealthImaging console exibe gráficos com base nos dados brutos da CloudWatch API.

Você deve ter as CloudWatch permissões apropriadas para monitorar HealthImaging CloudWatch. Para obter mais informações, consulte Gerenciamento de identidade e acesso CloudWatch no Guia do CloudWatch usuário.

HealthImaging Métricas da AWS

O `AWS/HealthImaging` namespace inclui as seguintes métricas nos níveis da conta e do armazenamento de dados.

Métricas em nível de conta

As métricas em nível de conta fornecem visibilidade agregada em todos os armazenamentos de dados em sua conta.

Métricas em nível de conta

Métrica	Description
`DataStoreCount`	O número de armazenamentos de dados com status ativo. Unidades: contagem Estatísticas válidas: soma, média
`ImageSetCount`	O número total de conjuntos de imagens em todos os armazenamentos de dados. Unidades: contagem Estatísticas válidas: soma, média
`StorageBytes`	A quantidade de dados em bytes armazenados em todos os armazenamentos de dados nas seguintes camadas de armazenamento: • Acesso frequente (`FrequentAccessStorage`) • Arquive o acesso instantâneo (`ArchiveInstantAccessStorage`) Esse valor é calculado pela soma do tamanho de todos os conjuntos de imagens em todos os armazenamentos de dados. Filtros de nível de armazenamento válidos (veja a `StorageTier` dimensão): • `FrequentAccessStorage` • `ArchiveInstantAccessStorage` • `AllStorage` Unidades: bytes

Métrica	Description
	Estatísticas válidas: soma, média

Métricas em nível de armazenamento de dados

As métricas em nível de armazenamento de dados fornecem visibilidade detalhada do armazenamento de dados individual.

Métricas em nível de armazenamento de dados

Métrica	Description
`TotalImageSetCount`	O número total de conjuntos de imagens no armazenamento de dados. Unidades: contagem Estatísticas válidas: soma, média
`PrimaryImageSetCount`	O número de conjuntos de imagens primárias no armazenamento de dados. Unidades: contagem Estatísticas válidas: soma, média
`SmallImageSetCount`	O número de conjuntos de imagens com menos de 5 MB no armazenamento de dados. Unidades: contagem Estatísticas válidas: soma, média
`StorageBytes`	A quantidade de dados em bytes armazenad os em todos os armazenamentos de dados nas seguintes camadas de armazenamento: • Acesso frequente (`FrequentAccessSto rage`)

Métrica	Description
	• Arquive o acesso instantâneo (`ArchiveInstantAccessStorage`) Esse valor é calculado pela soma do tamanho de todos os conjuntos de imagens no armazenamento de dados. Filtros de nível de armazenamento válidos (veja a `StorageTier` dimensão): • FrequentAccessStorage • ArchiveInstantAccessStorage • AllStorage Unidades: bytes Estatísticas válidas: soma, média
`DICOMStudyCount`	O número de estudos DICOM no armazenamento de dados. Unidades: contagem Estatísticas válidas: soma, média
`DICOMSeriesCount`	O número de séries DICOM no armazenamento de dados. Unidades: contagem Estatísticas válidas: soma, média

Métrica	Description
`DICOMInstanceCount`	O número de instâncias DICOM no armazenamento de dados. Unidades: contagem Estatísticas válidas: soma, média
`StructuredStorageBytes`	A quantidade de armazenamento estruturado em bytes indexada pelo armazenamento de dados. Unidades: bytes Estatísticas válidas: soma, média

HealthImaging Dimensões em CloudWatch

As dimensões a seguir são usadas para filtrar HealthImaging métricas.

Dimensões

Dimensão	Description
`AccountId`	Essa dimensão filtra os dados da AWS conta identificada.
`DatastoreId`	Essa dimensão filtra os dados somente para o armazenamento de dados identificado.
`StorageTier`	Essa dimensão filtra os dados pelas seguintes camadas de armazenamento: • `FrequentAccessStorage` — O número de bytes usados para conjuntos de imagens no armazenamento de acesso frequente. • `ArchiveInstantAccessStorage` — O número de bytes usados para conjuntos de

Dimensão	Description
	imagens no armazenamento Archive Instant Access. • `AllStorage` — O número total de bytes em todos os níveis de armazenamento.

Acessando HealthImaging métricas

Você pode acessar HealthImaging as métricas usando:

- Console de gerenciamento da AWS- Veja as métricas no CloudWatch console
- AWS CLI- Use comandos CloudWatch CLI
- CloudWatch API - Acesso por meio de AWS SDKs nossas ferramentas de CloudWatch API

Você deve ter as permissões apropriadas para monitorar HealthImaging CloudWatch. Para obter mais informações, consulte Gerenciamento de identidade e acesso CloudWatch no Guia do CloudWatch usuário.

Configurando HealthImaging métricas

Para receber HealthImaging métricas em sua CloudWatch conta, você precisa criar uma função vinculada ao serviço que permita HealthImaging publicar métricas em seu nome. Consulte Usando funções vinculadas a serviços HealthImaging para obter detalhes sobre como criar uma função vinculada a serviços.

Visualizando HealthImaging métricas

Para visualizar métricas (CloudWatch console)

1. Faça login no Console de gerenciamento da AWS e abra o CloudWatchconsole.
2. Escolha Métricas, escolha Todas as métricas e, em seguida, escolha **`AWS/HealthImaging`**.
3. Escolha a dimensão:
   - Por ID da conta - Exibir métricas em nível de conta
   - Por ID do Datastore - Visualize métricas em nível de armazenamento de dados
4. Escolha um nome de métrica e, em seguida, escolha Adicionar ao gráfico.

5. Escolha um valor para o intervalo de datas e a contagem métrica será exibida.

Criando um alarme usando CloudWatch

Um CloudWatch alarme monitora uma única métrica durante um período de tempo especificado e executa uma ou mais ações: enviar uma notificação para um tópico do Amazon Simple Notification Service (Amazon SNS) ou política de Auto Scaling. A ação ou ações são baseadas no valor da métrica em relação a um determinado limite em vários períodos que você especifica. CloudWatch também pode enviar uma mensagem do Amazon SNS quando o alarme muda de estado.

CloudWatch os alarmes invocam ações somente quando o estado muda e persiste pelo período especificado. Para obter mais informações, consulte Usando CloudWatch alarmes.

Usando a Amazon EventBridge com HealthImaging

EventBridge A Amazon é um serviço sem servidor que usa eventos para conectar componentes do aplicativo, facilitando a criação de aplicativos escaláveis orientados por eventos. A base do EventBridge é criar regras que direcionem eventos para os alvos. HealthImaging A AWS fornece entrega durável de mudanças de estado para EventBridge. Para obter mais informações, consulte O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon.

Tópicos

- HealthImaging eventos enviados para EventBridge
- HealthImaging estrutura e exemplos de eventos

HealthImaging eventos enviados para EventBridge

A tabela a seguir lista todos os HealthImaging eventos enviados EventBridge para processamento.

HealthImaging tipo de evento	Estado
Eventos do armazenamento de dados	
Criação de armazenamento de dados	`CREATING`
Falha na criação do armazenamento de dados	`CREATE_FAILED`

HealthImaging tipo de evento	Estado
Armazenamento de dados criado	`ACTIVE`
Exclusão do armazenamento de dados	`DELETING`
Armazenamento de dados excluído	`DELETED`
Para obter mais informações, consulte DatastoreStatus na AWS API Reference. HealthImaging	
Importar eventos de trabalho	
Import Job enviado	`SUBMITTED`
Importação de Job in Progress	`IN_PROGRESS`
Import Job concluído	`COMPLETED`
Falha no trabalho de importação	`FAILED`
Para obter mais informações, consulte JobStatus na AWS HealthImaging API Reference.	
Eventos do conjunto de imagens	
Conjunto de imagens criado	`CREATED`
Cópia do conjunto de imagens	`COPYING`
Cópia do conjunto de imagens com acesso somente para leitura	`COPYING_WITH_READ_ONLY_ACCESS`
Conjunto de imagens copiado	`COPIED`
Falha na cópia do conjunto de imagens	`COPY_FAILED`
Atualização do conjunto de imagens	`UPDATING`
Conjunto de imagens atualizado	`UPDATED`
Falha na atualização do conjunto de imagens	`UPDATE_FAILED`

HealthImaging tipo de evento	Estado
Exclusão do conjunto de imagens	`DELETING`
Conjunto de imagens excluído	`DELETED`

Para obter mais informações, consulte ImageSetWorkflowStatusa AWS HealthImaging API Reference.

HealthImaging estrutura e exemplos de eventos

HealthImaging eventos são objetos com estrutura JSON que também contêm detalhes de metadados. Você pode usar os metadados como entrada para recriar um evento ou obter mais informações. Todos os campos de metadados associados estão listados em uma tabela abaixo dos exemplos de código nos menus a seguir. Para obter mais informações, consulte Referência de estrutura de eventos no Guia EventBridge do usuário da Amazon.

> **Note**
>
> O `source` atributo para estruturas de HealthImaging eventos é`aws.medical-imaging`.

Eventos do armazenamento de dados

Data Store Creating

Estado - **`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

Estado - **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

Estado - **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

Estado - **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

Estado - **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"
    }
}
```

Eventos do armazenamento de dados - descrições de metadados

Nome	Tipo	Description
`version`	string	A versão EventBridge do esquema do evento.
`id`	string	O UUID da versão 4 gerado para cada evento.
`detail-type`	string	O tipo de evento que está sendo enviado.
`source`	string	Identifica o serviço que gerou o evento.
`account`	string	O ID de 12 dígitos da conta da AWS do proprietário do armazenamento de dados.
`time`	string	A hora em que o evento ocorreu.
`region`	string	Identifica a AWS região do armazenamento de dados.
`resources`	matriz (string)	Uma matriz JSON que contém o ARN do armazenamento de dados.
`detail`	objeto	Um objeto JSON contém informações sobre o evento.

Nome	Tipo	Description
`detail.imagingVersion`	string	O ID da versão que rastreia as alterações no esquema HealthImaging de detalhes do evento.
`detail.datastoreId`	string	O ID do armazenamento de dados associado ao evento de alteração de status.
`detail.datastoreName`	string	O nome do armazenamento de dados.
`detail.datastoreStatus`	string	O status atual do armazenamento de dados.

Importar eventos de trabalho

Import Job Submitted

Estado - **`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

Estado - **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

Estado - **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

Estado - **`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/"
    }
}
```

Importar eventos de trabalho - descrições de metadados

Nome	Tipo	Description
`version`	string	A versão EventBridge do esquema do evento.

Nome	Tipo	Description
`id`	string	O UUID da versão 4 gerado para cada evento.
`detail-type`	string	O tipo de evento que está sendo enviado.
`source`	string	Identifica o serviço que gerou o evento.
`account`	string	O ID de 12 dígitos da conta da AWS do proprietário do armazenamento de dados.
`time`	string	A hora em que o evento ocorreu.
`region`	string	Identifica a AWS região do armazenamento de dados.
`resources`	matriz (string)	Uma matriz JSON que contém o ARN do armazenamento de dados.
`detail`	objeto	Um objeto JSON contém informações sobre o evento.
`detail.imagingVersion`	string	O ID da versão que rastreia as alterações no esquema HealthImaging de detalhes do evento.
`detail.datastoreId`	string	O armazenamento de dados que gerou o evento de mudança de status.

Nome	Tipo	Description
`detail.jobId`	string	O ID do trabalho de importaçã o associado ao evento de mudança de status.
`detail.jobName`	string	O nome do trabalho de importação.
`detail.jobStatus`	string	O status atual do trabalho.
`detail.inputS3Uri`	string	O caminho do prefixo de entrada para o bucket do S3 que contém os arquivos DICOM a serem importados.
`detail.outputS3Uri`	string	O prefixo de saída do bucket S3 em que os resultados do trabalho de importação DICOM serão carregados.

Eventos do conjunto de imagens

Image Set Created

Estado - **`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

Estado - **COPYING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING",
      "sourceImageSetArn": "arn:aws:medical-imaging:us-
west-2:147997158357:datastore/c381ee9b9ef34902a45b476dd7be068b/
imageset/0309de3674fd551fa7ddd2880b21f990"
   }
}
```

Image Set Copying With Read Only Access

Estado - **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

Estado - **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

Estado - **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": "5b3a711878c34d40e8882533193886 49",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPY_FAILED"
   }
}
```

Image Set Updating

Estado - **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

Estado - **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

Estado - **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

Estado - **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

Estado - **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"
   }
}
```


Eventos do conjunto de imagens - descrições de metadados

Nome	Tipo	Description
`version`	string	A versão EventBridge do esquema do evento.
`id`	string	O UUID da versão 4 gerado para cada evento.
`detail-type`	string	O tipo de evento que está sendo enviado.
`source`	string	Identifica o serviço que gerou o evento.
`account`	string	O ID de 12 dígitos da conta da AWS do proprietário do armazenamento de dados.

Nome	Tipo	Description
`time`	string	A hora em que o evento ocorreu.
`region`	string	Identifica a AWS região do armazenamento de dados.
`resources`	matriz (string)	Uma matriz JSON que contém o ARN do conjunto de imagens.
`detail`	objeto	Um objeto JSON contém informações sobre o evento.
`detail.imagingVersion`	string	O ID da versão que rastreia as alterações no esquema HealthImaging de detalhes do evento.
`detail.isPrimary`	booleano	Indica se os dados importados foram organizados com êxito na hierarquia gerenciada ou se há conflitos de metadados que precisam ser resolvidos.
`detail.imageSetVersion`	string	A versão do conjunto de imagens será incrementada quando uma instância for importada mais de uma vez. A versão mais recente substituirá qualquer versão mais antiga armazenada em um conjunto de imagens primário.
`detail.datastoreId`	string	O ID do armazenamento de dados que gerou o evento de alteração de status.

Nome	Tipo	Description
`detail.imagesetId`	string	O ID do conjunto de imagens associado ao evento de alteração de status.
`detail.imageSetState`	string	O estado atual do conjunto de imagens.
`detail.imageSetWorkflowStatus`	string	O status atual do fluxo de trabalho do conjunto de imagens.

Segurança em AWS HealthImaging

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de data centers e arquiteturas de rede criados para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isso como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços no Nuvem AWS. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade que se aplicam AWS HealthImaging, consulte AWS Serviços no escopo do programa de conformidade AWS .
- Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Esta documentação ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao usar HealthImaging. Os tópicos a seguir mostram como configurar para atender HealthImaging aos seus objetivos de segurança e conformidade. Você também aprenderá a usar outros AWS serviços que ajudam a monitorar e proteger seus HealthImaging recursos.

Tópicos

- Proteção de dados na AWS HealthImaging
- Identity and Access Management para AWS HealthImaging
- Validação de conformidade para a AWS HealthImaging
- Segurança da infraestrutura na AWS HealthImaging
- Criação de HealthImaging recursos da AWS com AWS CloudFormation
- AWS HealthImaging e endpoints VPC de interface ()AWS PrivateLink
- Importação entre contas para AWS HealthImaging
- Resiliência na AWS HealthImaging

Proteção de dados na AWS HealthImaging

O modelo de responsabilidade AWS compartilhada de se aplica à proteção de dados na AWS HealthImaging. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para saber mais sobre a privacidade de dados, consulte as Data Privacy FAQ. Para saber mais sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and RGPD no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com Centro de Identidade do AWS IAM ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como trabalhar com CloudTrail trilhas no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sensíveis armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para saber mais sobre os endpoints FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sensíveis, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com HealthImaging ou Serviços da AWS usa o console, a API ou AWS SDKs. AWS CLI Quaisquer dados inseridos em tags ou em campos de texto de formato livre

usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, é fortemente recomendável que não sejam incluídas informações de credenciais no URL para validar a solicitação nesse servidor.

Tópicos

- Criptografia de dados
- Privacidade do tráfego de rede

Criptografia de dados

Com a AWS HealthImaging, você pode adicionar uma camada de segurança aos seus dados em repouso na nuvem, fornecendo recursos de criptografia escaláveis e eficientes. Isso inclui:

- Recursos de criptografia de dados em repouso disponíveis na maioria dos AWS serviços
- Opções flexíveis de gerenciamento de chaves AWS Key Management Service, inclusive, com as quais você pode escolher se deseja AWS gerenciar as chaves de criptografia ou manter o controle total sobre suas próprias chaves.
- AWS chaves de AWS KMS criptografia próprias
- Filas de mensagens criptografadas para a transmissão de dados confidenciais usando criptografia do lado do servidor (SSE) para o Amazon SQS

Além disso, AWS permite APIs que você integre criptografia e proteção de dados a qualquer um dos serviços que você desenvolve ou implanta em um AWS ambiente.

Criptografia em repouso

Por padrão, HealthImaging criptografa os dados do cliente em repouso usando uma chave de propriedade do serviço AWS Key Management Service . Opcionalmente, você pode configurar HealthImaging para criptografar dados em repouso usando uma AWS KMS chave simétrica gerenciada pelo cliente que você cria, possui e gerencia. Para obter mais informações, consulte Criar uma chave KMS de criptografia simétrica no Guia do AWS Key Management Service desenvolvedor.

Criptografia em trânsito

HealthImaging usa o TLS 1.2 para criptografar dados em trânsito pelo endpoint público e por meio de serviços de back-end.

Gerenciamento de chaves

AWS KMS chaves (chaves KMS) são o principal recurso em AWS Key Management Service. Você também pode gerar chaves de dados para uso fora do AWS KMS.

AWS chave KMS de propriedade

HealthImaging usa essas chaves por padrão para criptografar automaticamente informações potencialmente confidenciais, como dados de identificação pessoal ou dados de Informações de Saúde Privadas (PHI) em repouso. AWS as chaves KMS de propriedade não são armazenadas em sua conta. Elas fazem parte de uma coleção de chaves KMS que AWS possui e gerencia para uso em várias AWS contas. AWS os serviços podem usar chaves KMS AWS próprias para proteger seus dados. Você não pode visualizar, gerenciar, usar chaves KMS AWS próprias nem auditar seu uso. No entanto, você não precisa fazer nenhum trabalho nem alterar nenhum programa para proteger as chaves que criptografam seus dados.

Não é cobrada uma taxa mensal ou uma taxa de uso se você usar chaves KMS AWS próprias, e elas não contam nas AWS KMS cotas da sua conta. Para obter mais informações, consulte Chaves de propriedade da AWS no Guia do desenvolvedor do AWS Key Management Service .

Chaves do KMS gerenciadas pelo cliente

Se você quiser controle total sobre o AWS KMS ciclo de vida e o uso, HealthImaging suporta o uso de uma chave KMS simétrica gerenciada pelo cliente que você cria, possui e gerencia. Como você tem controle total dessa camada de criptografia, você pode realizar tarefas como:

- Estabelecer e manter políticas de chaves, políticas do IAM e concessões
- Rotacionar os materiais de chave de criptografia
- Habilitar e desabilitar políticas de chaves
- Adicionar etiquetas
- Criar réplicas de chaves
- Chaves de agendamento para exclusão

Você também pode usar CloudTrail para rastrear as solicitações HealthImaging enviadas AWS KMS em seu nome. AWS KMS Taxas adicionais se aplicam. Para obter mais informações, consulte Chaves gerenciadas pelo cliente no Guia do AWS Key Management Service desenvolvedor.

Criar uma chave gerenciada pelo cliente

Você pode criar uma chave simétrica gerenciada pelo cliente usando o Console de gerenciamento da AWS ou o. AWS KMS APIs Para obter mais informações, consulte Criar chaves do KMS simétricas no Guia do desenvolvedor do AWS Key Management Service .

As políticas de chaves controlam o acesso à chave gerenciada pelo cliente. Cada chave gerenciada pelo cliente deve ter exatamente uma política de chaves, que contém declarações que determinam quem pode usar a chave e como pode usá-la. Ao criar a chave gerenciada pelo cliente, você pode especificar uma política de chaves. Para obter mais informações, consulte Gerenciamento do acesso às chaves gerenciadas pelo cliente no Guia do desenvolvedor do AWS Key Management Service .

Para usar sua chave gerenciada pelo cliente com seus HealthImaging recursos, CreateGrant as operações kms: devem ser permitidas na política de chaves. Isso adiciona uma concessão a uma chave gerenciada pelo cliente que controla o acesso a uma chave KMS especificada, que dá ao usuário acesso às operações de concessão HealthImaging necessárias. Para obter mais informações, consulte Concessões no AWS KMS no Guia do desenvolvedor do AWS Key Management Service .

Para usar sua chave KMS gerenciada pelo cliente com seus HealthImaging recursos, as seguintes operações de API devem ser permitidas na política de chaves:

- `kms:DescribeKey` fornece os principais detalhes gerenciados pelo cliente necessários para validar a chave. Isso é necessário para todas as operações.
- `kms:GenerateDataKey` fornece acesso para criptografar recursos em repouso para todas as operações de gravação.
- `kms:Decrypt` fornece acesso às operações de leitura ou pesquisa de recursos criptografados.
- `kms:ReEncrypt*` fornece acesso para recriptografar recursos.

Veja a seguir um exemplo de declaração de política que permite que um usuário crie e interaja com um armazenamento de HealthImaging dados criptografado por essa chave:

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

Permissões do IAM obrigatórias para usar uma chave KMS gerenciada pelo cliente

Ao criar um armazenamento de dados com AWS KMS criptografia habilitada usando uma chave KMS gerenciada pelo cliente, há permissões necessárias para a política de chaves e a política do IAM para o usuário ou função que está criando o armazenamento de HealthImaging dados.

Para obter mais informações sobre políticas de chave, consulte Habilitar políticas do IAM no Guia do desenvolvedor do AWS Key Management Service .

O usuário do IAM, a função do IAM ou a AWS conta que está criando seus repositórios devem ter permissões para a política abaixo, além das permissões necessárias para a AWS HealthImaging.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:GenerateDataKey",
        "kms:RetireGrant",
        "kms:Decrypt",
```

```
        "kms:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f"
    }
  ]
}
```

Como HealthImaging usa subsídios em AWS KMS

HealthImaging exige uma concessão para usar sua chave KMS gerenciada pelo cliente. Quando você cria um armazenamento de dados criptografado com uma chave KMS gerenciada pelo cliente, HealthImaging cria uma concessão em seu nome enviando uma CreateGrantsolicitação para AWS KMS. As concessões AWS KMS são usadas para dar HealthImaging acesso a uma chave KMS em uma conta de cliente.

Os subsídios HealthImaging criados em seu nome não devem ser revogados ou retirados. Se você revogar ou retirar a concessão que dá HealthImaging permissão para usar as AWS KMS chaves em sua conta, HealthImaging não conseguir acessar esses dados, criptografar novos recursos de imagem enviados ao armazenamento de dados ou descriptografá-los quando forem retirados. Quando você revoga ou retira um subsídio HealthImaging, a alteração ocorre imediatamente. Para revogar direitos de acesso, você deve excluir o datastore em vez de revogar a concessão. Quando um armazenamento de dados é excluído, HealthImaging as concessões são retiradas em seu nome.

Monitorando suas chaves de criptografia para HealthImaging

Você pode usar CloudTrail para rastrear as solicitações HealthImaging enviadas AWS KMS em seu nome ao usar uma chave KMS gerenciada pelo cliente. As entradas de registro no CloudTrail registro são exibidas `medical-imaging.amazonaws.com` no `userAgent` campo para distinguir claramente as solicitações feitas por HealthImaging.

Os exemplos a seguir são CloudTrail eventos para`CreateGrant`, `GenerateDataKeyDecrypt`, e `DescribeKey` para monitorar AWS KMS operações chamadas por HealthImaging para acessar dados criptografados pela chave gerenciada pelo cliente.

A seguir, mostramos como usar `CreateGrant` para permitir HealthImaging o acesso à chave KMS fornecida pelo cliente, permitindo usar essa chave KMS HealthImaging para criptografar todos os dados do cliente em repouso.

Os usuários não precisam criar suas próprias concessões. HealthImaging cria uma concessão em seu nome enviando uma `CreateGrant` solicitação para AWS KMS. As concessões AWS KMS são usadas para dar HealthImaging acesso a uma AWS KMS chave na conta de um cliente.


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

Os exemplos a seguir mostram como usar `GenerateDataKey` para garantir que o usuário tenha as permissões necessárias para criptografar dados antes de armazená-los.

```
{
    "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"
}
```

O exemplo a seguir mostra como HealthImaging chama a `Decrypt` operação para usar a chave de dados criptografada armazenada para acessar os dados criptografados.


```
{
    "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"
}
```

O exemplo a seguir mostra como HealthImaging usa a `DescribeKey` operação para verificar se a AWS KMS chave de propriedade do AWS KMS cliente está em um estado utilizável e para ajudar o usuário a solucionar problemas se ela não estiver funcionando.

```
{
    "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"
}
```


Saiba mais

Os recursos a seguir fornecem mais informações sobre criptografia de dados em repouso e estão localizados no Guia do desenvolvedor do AWS Key Management Service .

- Conceitos do AWS KMS
- Melhores práticas de segurança para AWS KMS

Privacidade do tráfego de rede

O tráfego é protegido entre aplicativos locais HealthImaging HealthImaging e entre o Amazon S3. O tráfego entre HealthImaging e AWS Key Management Service usa HTTPS por padrão.

- HealthImaging A AWS é um serviço regional disponível nas regiões Leste dos EUA (Norte da Virgínia), Oeste dos EUA (Oregon), Europa (Irlanda) e Ásia-Pacífico (Sydney).
- Para tráfego entre os HealthImaging buckets do Amazon S3, o Transport Layer Security (TLS) criptografa objetos em trânsito entre o HealthImaging Amazon S3 e entre os aplicativos do cliente que o acessam, você deve permitir somente conexões criptografadas por HTTPS (TLS) usando as

políticas do IAM do bucket do `aws:SecureTransport condition`Amazon S3. HealthImaging Embora HealthImaging atualmente use o endpoint público para acessar dados nos buckets do Amazon S3, isso não significa que os dados atravessem a Internet pública. Todo o tráfego entre o Amazon S3 HealthImaging e o Amazon S3 é roteado pela AWS rede e criptografado usando TLS.

Identity and Access Management para AWS HealthImaging

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar HealthImaging os recursos. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Tópicos

- Público
- Autenticação com identidades
- Gerenciar o acesso usando políticas
- Como a AWS HealthImaging trabalha com o IAM
- Exemplos de políticas baseadas em identidade para a AWS HealthImaging
- AWS políticas gerenciadas para a AWS HealthImaging
- Prevenção confusa de delegados entre serviços em HealthImaging
- Usando funções vinculadas a serviços para HealthImaging
- Solução de problemas de HealthImaging identidade e acesso à AWS

Público

A forma como você usa AWS Identity and Access Management (IAM) difere com base na sua função:

- Usuário do serviço: solicite permissões ao seu administrador se você não conseguir acessar os atributos (consulte Solução de problemas de HealthImaging identidade e acesso à AWS).
- Administrador do serviço: determine o acesso do usuário e envie solicitações de permissão (consulte Como a AWS HealthImaging trabalha com o IAM)

- Administrador do IAM: escreva políticas para gerenciar o acesso (consulte Exemplos de políticas baseadas em identidade para a AWS HealthImaging)

Autenticação com identidades

A autenticação é como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado como usuário do IAM ou assumindo uma função do IAM. Usuário raiz da conta da AWS

Você pode fazer login como uma identidade federada usando credenciais de uma fonte de identidade como Centro de Identidade do AWS IAM (IAM Identity Center), autenticação de login único ou credenciais. Google/Facebook Para ter mais informações sobre como fazer login, consulte Como fazer login em sua Conta da AWS no Guia do usuário do Início de Sessão da AWS .

Para acesso programático, AWS fornece um SDK e uma CLI para assinar solicitações criptograficamente. Para ter mais informações, consulte AWS Signature Version 4 para solicitações de API no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar um Conta da AWS, você começa com uma identidade de login chamada usuário Conta da AWS raiz que tem acesso completo a todos Serviços da AWS os recursos. É altamente recomendável não usar o usuário-raiz em tarefas diárias. Consulte as tarefas que exigem credenciais de usuário-raiz em Tarefas que exigem credenciais de usuário-raiz no Guia do usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários humanos usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório corporativo, provedor de identidade da web ou Directory Service que acessa Serviços da AWS usando credenciais de uma fonte de identidade. As identidades federadas assumem funções que oferecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos Centro de Identidade do AWS IAM. Para saber mais, consulte O que é o IAM Identity Center? no Guia do usuário do Centro de Identidade do AWS IAM .

Usuários e grupos do IAM

Um usuário do IAM é uma identidade com permissões específicas para uma única pessoa ou aplicação. É recomendável usar credenciais temporárias, em vez de usuários do IAM com credenciais de longo prazo. Para obter mais informações, consulte Exigir que usuários humanos usem a federação com um provedor de identidade para acessar AWS usando credenciais temporárias no Guia do usuário do IAM.

Um grupo do IAM especifica um conjunto de usuários do IAM e facilita o gerenciamento de permissões para grandes conjuntos de usuários. Para ter mais informações, consulte Casos de uso de usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Uma perfil do IAM é uma identidade com permissões específicas que oferece credenciais temporárias. Você pode assumir uma função mudando de um usuário para uma função do IAM (console) ou chamando uma operação de AWS API AWS CLI ou. Para saber mais, consulte Métodos para assumir um perfil no Manual do usuário do IAM.

Os perfis do IAM são úteis para acesso de usuário federado, permissões de usuário do IAM temporárias, acesso entre contas, acesso entre serviços e aplicações em execução no Amazon EC2. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política define permissões quando associada a uma identidade ou recurso. AWS avalia essas políticas quando um diretor faz uma solicitação. A maioria das políticas é armazenada AWS como documentos JSON. Para ter mais informações sobre documentos de política JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Por meio de políticas, os administradores especificam quem tem acesso a que, definindo qual entidade principal pode realizar ações em quais recursos e sob quais condições.

Por padrão, usuários e perfis não têm permissões. Um administrador do IAM cria políticas do IAM e as adiciona aos perfis, os quais os usuários podem então assumir. As políticas do IAM definem permissões, independentemente do método usado para realizar a operação.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissão JSON que você anexa a uma identidade (usuário, grupo ou perfil). Essas políticas controlam quais ações as identidades podem realizar, em quais recursos e sob quais condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser políticas em linha (incorporadas diretamente em uma única identidade) ou políticas gerenciadas (políticas autônomas anexadas a várias identidades). Para saber como escolher entre uma política gerenciada e políticas em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. Entre os exemplos estão políticas de confiança de perfil do IAM e políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. É necessário especificar uma entidade principal em uma política baseada em recursos.

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas do IAM em uma política baseada em recursos.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais que podem definir o máximo de permissões concedidas por tipos de políticas mais comuns:

- Limites de permissões: definem o número máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM. Para saber mais sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs) — Especifique as permissões máximas para uma organização ou unidade organizacional em AWS Organizations. Para saber mais, consulte Políticas de controle de serviço no Guia do usuário do AWS Organizations .
- Políticas de controle de recursos (RCPs) — Defina o máximo de permissões disponíveis para recursos em suas contas. Para obter mais informações, consulte Políticas de controle de recursos (RCPs) no Guia AWS Organizations do usuário.

- Políticas de sessão: políticas avançadas transmitidas como um parâmetro durante a criação de uma sessão temporária para um perfil ou um usuário federado. Para saber mais, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

Como a AWS HealthImaging trabalha com o IAM

Antes de usar o IAM para gerenciar o acesso HealthImaging, saiba com quais recursos do IAM estão disponíveis para uso HealthImaging.

Recursos do IAM que você pode usar com a AWS HealthImaging

Recurso do IAM	HealthImaging apoio
Políticas baseadas em identidade	Sim
Políticas baseadas em recurso	Não
Ações de políticas	Sim
Recursos de políticas	Sim
Chaves de condição de política (específicas do serviço)	Sim
ACLs	Não
ABAC (tags em políticas)	Parcial
Credenciais temporárias	Sim
Permissões de entidade principal	Sim

Recurso do IAM	HealthImaging apoio
Perfis de serviço	Sim
Perfis vinculados a serviço	Não

Para ter uma visão de alto nível de como HealthImaging e outros AWS serviços funcionam com a maioria dos recursos do IAM, consulte AWS os serviços que funcionam com o IAM no Guia do usuário do IAM.

Políticas baseadas em identidade para HealthImaging

Compatível com políticas baseadas em identidade: sim

As políticas baseadas em identidade são documentos de políticas de permissões JSON que podem ser anexados a uma identidade, como usuário do IAM, grupo de usuários ou perfil. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

Com as políticas baseadas em identidade do IAM, é possível especificar ações e recursos permitidos ou negados, assim como as condições sob as quais as ações são permitidas ou negadas. Para saber mais sobre todos os elementos que podem ser usados em uma política JSON, consulte Referência de elemento de política JSON do IAM no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade para HealthImaging

Para ver exemplos de políticas HealthImaging baseadas em identidade, consulte. Exemplos de políticas baseadas em identidade para a AWS HealthImaging

Políticas baseadas em recursos dentro HealthImaging

Compatibilidade com políticas baseadas em recursos: não

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico.

Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. É necessário especificar uma entidade principal em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Para permitir o acesso entre contas, é possível especificar uma conta inteira ou as entidades do IAM em outra conta como a entidade principal em uma política baseada em recursos. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Ações políticas para HealthImaging

Compatível com ações de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. Incluem ações em uma política para conceder permissões para executar a operação associada.

Para ver uma lista de HealthImaging ações, consulte Ações definidas pela AWS HealthImaging na Referência de autorização de serviço.

As ações de política HealthImaging usam o seguinte prefixo antes da ação:

```
AWS
```

Para especificar várias ações em uma única declaração, separe-as com vírgulas.

```
"Action": [
      "AWS:action1",
      "AWS:action2"
         ]
```

Para ver exemplos de políticas HealthImaging baseadas em identidade, consulte. Exemplos de políticas baseadas em identidade para a AWS HealthImaging

Recursos políticos para HealthImaging

Compatível com recursos de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon (ARN). Para ações que não oferecem compatibilidade com permissões em nível de recurso, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

Para ver uma lista dos tipos de HealthImaging recursos e seus ARNs, consulte Tipos de recursos definidos pela AWS HealthImaging na Referência de Autorização de Serviços. Para saber com quais ações e recursos você pode usar um ARN, consulte Ações definidas pela AWS. HealthImaging

Para ver exemplos de políticas HealthImaging baseadas em identidade, consulte. Exemplos de políticas baseadas em identidade para a AWS HealthImaging

Chaves de condição de política para HealthImaging

Compatível com chaves de condição de política específicas de serviço: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` especifica quando as instruções são executadas com base em critérios definidos. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação. Para ver todas as chaves de condição AWS globais, consulte as chaves de contexto de condição AWS global no Guia do usuário do IAM.

Para ver uma lista de chaves de HealthImaging condição, consulte Chaves de condição para AWS HealthImaging na Referência de autorização de serviço. Para saber com quais ações e recursos você pode usar uma chave de condição, consulte Ações definidas pela AWS HealthImaging.

Para ver exemplos de políticas HealthImaging baseadas em identidade, consulte. Exemplos de políticas baseadas em identidade para a AWS HealthImaging

ACLs in HealthImaging

Suportes ACLs: Não

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

RBAC com HealthImaging

É compatível com o RBAC	Sim

O modelo de autorização tradicional usado no IAM é chamado de controle de acesso baseado em função (RBAC). O RBAC define permissões com base na função de trabalho de uma pessoa, conhecida fora da AWS como uma função. Para obter mais informações, consulte Comparar o ABAC com o modelo de RBAC tradicional no Guia do usuário do IAM.

ABAC com HealthImaging

Compatível com ABAC (tags em políticas): parcial

> **Warning**
>
> O ABAC não é aplicado por meio de ação `SearchImageSets` da API. Qualquer pessoa que tenha acesso à ação `SearchImageSets` pode acessar todos os metadados dos conjuntos de imagens em um datastore.

> **Note**
>
> Os conjuntos de imagens são um recurso secundário dos datastores. Para usar o ABAC, um conjunto de imagens deve ter a mesma tag de um datastore. Para obter mais informações, consulte Marcação de recursos com a AWS HealthImaging.

O controle de acesso por atributo (ABAC) é uma estratégia de autorização que define permissões com base em atributos chamados de tags. Você pode anexar tags a entidades e AWS recursos

do IAM e, em seguida, criar políticas ABAC para permitir operações quando a tag do diretor corresponder à tag no recurso.

Para controlar o acesso baseado em tags, forneça informações sobre as tags no elemento de condição de uma política usando as `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou chaves de condição `aws:TagKeys`.

Se um serviço for compatível com as três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço for compatível com as três chaves de condição somente para alguns tipos de recursos, o valor será Parcial

Para saber mais sobre o ABAC, consulte Definir permissões com autorização do ABAC no Guia do usuário do IAM. Para visualizar um tutorial com etapas para configurar o ABAC, consulte Usar controle de acesso por atributo (ABAC) no Guia do usuário do IAM.

Usando credenciais temporárias com HealthImaging

Compatível com credenciais temporárias: sim

As credenciais temporárias fornecem acesso de curto prazo aos AWS recursos e são criadas automaticamente quando você usa a federação ou troca de funções. AWS recomenda que você gere credenciais temporárias dinamicamente em vez de usar chaves de acesso de longo prazo. Para ter mais informações, consulte Credenciais de segurança temporárias no IAM e Serviços da Serviços da AWS que funcionam com o IAM no Guia do usuário do IAM.

Permissões principais entre serviços para HealthImaging

Compatibilidade com o recurso de encaminhamento de sessões de acesso (FAS): sim

Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado um principal. Permissões concedidas por políticas a uma entidade principal. Quando você usa alguns serviços, pode executar uma ação que, em seguida, acionar outra ação em outro serviço. Nesse caso, você precisa ter permissões para executar ambas as ações. Para ver se uma ação exige ações dependentes adicionais em uma política, consulte Ações, recursos e chaves de condição para a AWS HealthImaging na Referência de autorização de serviço.

Funções de serviço para HealthImaging

Compatível com perfis de serviço: sim

O perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para saber

mais, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.

> **Warning**
>
> Alterar as permissões de uma função de serviço pode interromper HealthImaging a funcionalidade. Edite as funções de serviço somente quando HealthImaging fornecer orientação para fazer isso.

Funções vinculadas a serviços para HealthImaging

Compatível com perfis vinculados ao serviço: Não

Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode assumir o perfil de executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados ao serviço.

Para obter detalhes sobre como criar ou gerenciar perfis vinculados a serviços, consulte Serviços da AWS que funcionam com o IAM. Encontre um serviço na tabela que inclua um `Yes` na coluna Perfil vinculado ao serviço. Escolha o link Sim para visualizar a documentação do perfil vinculado a serviço desse serviço.

Exemplos de políticas baseadas em identidade para a AWS HealthImaging

Por padrão, usuários e perfis não têm permissão para criar ou modificar recursos do HealthImaging. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM.

Para aprender a criar uma política baseada em identidade do IAM ao usar esses documentos de política em JSON de exemplo, consulte Criar políticas do IAM (console) no Guia do usuário do IAM.

Para obter detalhes sobre ações e tipos de recursos definidos pelo Awesome, incluindo o formato do ARNs para cada um dos tipos de recursos, consulte Ações, recursos e chaves de condição do AWS Awesome na Referência de autorização de serviço.

Tópicos

- Práticas recomendadas de política

- Usando o HealthImaging console
- Permitir que os usuários visualizem suas próprias permissões

Práticas recomendadas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir HealthImaging recursos em sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas AWS gerenciadas e avance para as permissões de privilégios mínimos — Para começar a conceder permissões aos seus usuários e cargas de trabalho, use as políticas AWS gerenciadas que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis no seu Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo AWS cliente que sejam específicas para seus casos de uso. Para saber mais, consulte Políticas gerenciadas pela AWS ou Políticas gerenciadas pela AWS para funções de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para saber mais sobre como usar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.
- Use condições nas políticas do IAM para restringir ainda mais o acesso: é possível adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, é possível escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica AWS service (Serviço da AWS), como CloudFormation. Para saber mais, consulte Elementos da política JSON do IAM: condição no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para saber mais, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.
- Exigir autenticação multifator (MFA) — Se você tiver um cenário que exija usuários do IAM ou um usuário root, ative Conta da AWS a MFA para obter segurança adicional. Para exigir MFA quando

as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para saber mais, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para saber mais sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Usando o HealthImaging console

Para acessar o HealthImaging console da AWS, você deve ter um conjunto mínimo de permissões. Essas permissões devem permitir que você liste e visualize detalhes sobre os HealthImaging recursos em seu Conta da AWS. Caso crie uma política baseada em identidade mais restritiva que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou perfis) com essa política.

Você não precisa permitir permissões mínimas do console para usuários que estão fazendo chamadas somente para a API AWS CLI ou para a AWS API. Em vez disso, permita o acesso somente a ações que correspondam à operação de API que estiverem tentando executar.

Para garantir que usuários e funções ainda possam usar o HealthImaging console, anexe também a política HealthImaging *ConsoleAccess* ou a política *ReadOnly* AWS gerenciada às entidades. Para obter informações, consulte Adicionar permissões a um usuário no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou programaticamente usando a API AWS CLI ou AWS .

```
{
    "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 políticas gerenciadas para a AWS HealthImaging

Uma política AWS gerenciada é uma política autônoma criada e administrada por AWS. AWS as políticas gerenciadas são projetadas para fornecer permissões para muitos casos de uso comuns, para que você possa começar a atribuir permissões a usuários, grupos e funções.

Lembre-se de que as políticas AWS gerenciadas podem não conceder permissões de privilégio mínimo para seus casos de uso específicos porque elas estão disponíveis para uso de todos os AWS clientes. Recomendamos que você reduza ainda mais as permissões definindo as políticas gerenciadas pelo cliente que são específicas para seus casos de uso.

Você não pode alterar as permissões definidas nas políticas AWS gerenciadas. Se AWS atualizar as permissões definidas em uma política AWS gerenciada, a atualização afetará todas as identidades principais (usuários, grupos e funções) às quais a política está anexada. AWS é mais provável que atualize uma política AWS gerenciada quando uma nova AWS service (Serviço da AWS) é lançada ou novas operações de API são disponibilizadas para serviços existentes.

Para saber mais, consulte AWS Políticas gerenciadas pela no Guia do usuário do IAM.

Tópicos

- AWS política gerenciada: AWSHealth ImagingServiceRolePolicy
- AWS política gerenciada: AWSHealth ImagingFullAccess
- AWS política gerenciada: AWSHealth ImagingReadOnlyAccess
- HealthImaging atualizações nas políticas AWS gerenciadas

AWS política gerenciada: AWSHealth ImagingServiceRolePolicy

Essa política está vinculada à função vinculada ao serviço. `AWSServiceRoleForHealthImaging` Ele concede permissões HealthImaging para gerenciar operações de serviço e publicar métricas de serviço.

Para obter mais informações sobre essa política, incluindo o documento de política JSON, consulte AWSHealthImagingServiceRolePolicyo AWS Managed Policy Reference Guide.

AWS política gerenciada: AWSHealth ImagingFullAccess

É possível anexar a política `AWSHealthImagingFullAccess` às suas identidades do IAM.

Essa política concede permissão administrativa para todas as HealthImaging ações.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "medical-imaging:*"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "medical-imaging.amazonaws.com"
            }
        }
        }
    ]
}
```

AWS política gerenciada: AWSHealth ImagingReadOnlyAccess

É possível anexar a política `AWSHealthImagingReadOnlyAccess` às suas identidades do IAM.

Essa política concede permissão somente de leitura para ações específicas da AWS HealthImaging .

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "medical-imaging:GetDICOMImportJob",
            "medical-imaging:GetDatastore",
            "medical-imaging:GetImageFrame",
            "medical-imaging:GetImageSet",
            "medical-imaging:GetImageSetMetadata",
            "medical-imaging:ListDICOMImportJobs",
            "medical-imaging:ListDatastores",
            "medical-imaging:ListImageSetVersions",
            "medical-imaging:ListTagsForResource",
            "medical-imaging:SearchImageSets"
        ],
```

```
        "Resource": "*"
    }]
}
```

HealthImaging atualizações nas políticas AWS gerenciadas

Veja detalhes sobre as atualizações das políticas AWS gerenciadas HealthImaging desde que esse serviço começou a rastrear essas alterações. Para obter alertas automáticos sobre alterações feitas nesta página, inscreva-se no feed RSS na página Lançamentos.

Alteração	Descrição	Data
AWSHealthImagingServiceRolePolicy	A AWS HealthImaging adicionou uma nova política gerenciada para a função vinculada ao serviço que fornece permissões HealthImaging para gerenciar operações de serviço e publicar métricas de serviço.	9 de fevereiro de 2026
HealthImaging começou a rastrear alterações	HealthImaging começou a rastrear as mudanças em suas políticas AWS gerenciadas.	19 de julho de 2023

Prevenção confusa de delegados entre serviços em HealthImaging

O problema "confused deputy" é um problema de segurança em que uma entidade que não tem permissão para executar uma ação pode coagir uma entidade mais privilegiada a executar a ação. Na AWS, a falsificação de identidade entre serviços pode resultar em um problema confuso de delegado. A personificação entre serviços pode ocorrer quando um serviço (o serviço de chamada) chama outro serviço (o serviço chamado). O serviço de chamada pode ser manipulado de modo

a usar suas permissões para atuar nos recursos de outro cliente de uma forma na qual ele não deveria ter permissão para acessar. Para evitar isso, a AWS fornece ferramentas que ajudam você a proteger seus dados para todos os serviços com diretores de serviços que receberam acesso aos recursos em sua conta.

Recomendamos usar as `aws:SourceArn`chaves de contexto de condição `aws:SourceAccount`global em suas políticas de relacionamento de confiança da função do `ImportJobDataAccessRole` IAM para limitar as permissões que a AWS HealthImaging concede a outro serviço ao seu recurso. Use `aws:SourceArn` se quiser associar apenas um recurso ao acesso entre serviços. Use `aws:SourceAccount` se quiser permitir que qualquer recurso nessa conta seja associado ao uso entre serviços. Se você usar ambas as chaves de contexto de condição global, o valor `aws:SourceAccount` e a conta referenciada no valor `aws:SourceArn` deverão usar o mesmo ID de conta quando usados na mesma declaração de política.

O valor de `aws:SourceArn` deve ser o ARN do armazenamento de dados afetado. Se você não souber o ARN completo do armazenamento de dados ou se estiver especificando vários armazenamentos de dados, use a chave de condição de contexto `aws:SourceArn` global com o caractere curinga * para as partes desconhecidas do ARN. Por exemplo, você pode usar o `aws:SourceArn` para `arn:aws:medical-imaging:us-west-2:111122223333:datastore/*`.

No exemplo de política de confiança a seguir, usamos a chave de `aws:SourceAccount` condição `aws:SourceArn` e para restringir o acesso ao principal do serviço com base no ARN do armazenamento de dados para evitar o problema confuso do substituto.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "medical-imaging.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/*"
```

```
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Usando funções vinculadas a serviços para HealthImaging

AWS HealthImaging usa funções vinculadas ao serviço AWS Identity and Access Management (IAM) que são predefinidas pelo serviço e incluem todas as permissões que o serviço exige para chamar outros serviços da AWS em seu nome. Para obter mais informações, consulte Permissões de perfil vinculado ao serviço no Guia do usuário do IAM.

Permissões de função vinculadas ao serviço para HealthImaging

HealthImaging usa a função vinculada ao serviço nomeada `AWSServiceRoleForHealthImaging` para realizar operações em sua AWS conta. Você precisa criar essa função vinculada ao serviço se quiser HealthImaging publicar métricas sobre seus armazenamentos de dados no. CloudWatch

A política de permissões de função denominada `AWSHealthImagingServiceRolePolicy` concede permissões HealthImaging para gerenciar operações de serviço e publicar métricas de serviço.

Para atualizações de políticas gerenciadas, consulte políticas HealthImaging gerenciadas.

Criação de uma função vinculada ao serviço para HealthImaging

Crie uma função vinculada ao serviço com o console do IAM

Você pode criar uma função vinculada ao serviço usando o console do IAM selecionando `AWS Service` como o tipo de entidade confiável e, `HealthImaging` em seguida, no menu suspenso Caso de uso.

Crie uma função vinculada ao serviço com a AWS CLI

Na AWS CLI, execute `aws iam create-service-linked-role –aws-service-name medical-imaging.amazonaws.com`

Excluindo uma função vinculada ao serviço para HealthImaging

Você pode excluir uma função vinculada ao serviço a qualquer momento, mas isso HealthImaging impedirá a execução de ações em sua AWS conta, como a publicação de métricas do armazenamento de dados em. CloudWatch

Como excluir manualmente o perfil vinculado ao serviço usando o IAM

Você pode usar o console do IAM, a AWS CLI ou a API da AWS para excluir a função vinculada ao `AWSServiceRoleForHealthImaging` serviço. Para saber mais, consulte Excluir um perfil vinculado ao serviço no Guia do usuário do IAM. Se você excluiu uma função vinculada ao serviço, poderá usar o processo de criação da função para criar uma nova.

Solução de problemas de HealthImaging identidade e acesso à AWS

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com HealthImaging um IAM.

Tópicos

- Não estou autorizado a realizar uma ação em HealthImaging
- Não estou autorizado a realizar iam: PassRole
- Quero permitir que pessoas fora da minha Conta da AWS acessem meus HealthImaging recursos

Não estou autorizado a realizar uma ação em HealthImaging

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um atributo *`my-example-widget`* fictício, mas não tem as permissões `AWS:`*`GetWidget`* fictícias.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 AWS:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso *`my-example-widget`* usando a ação `AWS:`*`GetWidget`*.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a realizar iam: PassRole

Se você receber uma mensagem de erro informando que não está autorizado a executar a ação `iam:PassRole`, as suas políticas devem ser atualizadas para permitir que você passe uma função para o HealthImaging.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazê-lo, você deve ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta utilizar o console para executar uma ação no HealthImaging. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas fora da minha Conta da AWS acessem meus HealthImaging recursos

É possível criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se é HealthImaging compatível com esses recursos, consulteComo a AWS HealthImaging trabalha com o IAM.

- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Como fornecer acesso a um usuário do IAM em outro Conta da AWS que você possui no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como fornecer acesso Contas da AWS a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Validação de conformidade para a AWS HealthImaging

Auditores terceirizados avaliam a segurança e a conformidade da AWS HealthImaging como parte de vários programas de AWS conformidade. Pois HealthImaging, isso inclui a HIPAA.

Para obter uma lista de AWS serviços no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo do programa de conformidade . Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact.

Sua responsabilidade de conformidade ao usar a AWS HealthImaging é determinada pela confidencialidade dos seus dados, pelos objetivos de conformidade da sua empresa e pelas leis e regulamentações aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- AWS Soluções de parceiros — Os guias automatizados de implantação de referência para segurança e conformidade discutem considerações arquitetônicas e fornecem etapas para a implantação de ambientes básicos focados em segurança e conformidade em. AWS
- Documento técnico sobre arquitetura para segurança e conformidade com a HIPAA — Este whitepaper descreve como as empresas podem usar para criar aplicativos compatíveis com a HIPAA. AWS
- GxP Systems on AWS — Este whitepaper fornece informações sobre como AWS abordar a conformidade e a segurança relacionadas ao GxP e fornece orientação sobre o uso de AWS serviços no contexto do GxP.

- AWS Recursos de conformidade — Essa coleção de pastas de trabalho e guias pode ser aplicada ao seu setor e local.
- Avaliação de recursos com regras — AWS Config avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes do setor e os regulamentos.
- AWS Security Hub CSPM— Esse AWS serviço fornece uma visão abrangente do seu estado de segurança interno, AWS que ajuda você a verificar sua conformidade com os padrões e as melhores práticas do setor de segurança.

Segurança da infraestrutura na AWS HealthImaging

Como um serviço gerenciado, a AWS HealthImaging é protegida pelos procedimentos AWS globais de segurança de rede descritos no whitepaper Amazon Web Services: Visão geral dos processos de segurança.

Você usa chamadas de API AWS publicadas para acessar HealthImaging pela rede. Os clientes devem oferecer suporte a Transport Layer Security (TLS) 1.3 ou posterior. Os clientes também devem ter compatibilidade com conjuntos de criptografia com perfect forward secrecy (PFS) como Ephemeral Diffie-Hellman (DHE) ou Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). A maioria dos sistemas modernos como Java 7 e versões posteriores oferece suporte a esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Também é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Criação de HealthImaging recursos da AWS com AWS CloudFormation

HealthImaging A AWS está integrada com AWS CloudFormation um serviço que ajuda você a modelar e configurar seus AWS recursos para que você possa gastar menos tempo criando e gerenciando seus recursos e infraestrutura. Você cria um modelo que descreve todos os AWS recursos que você deseja, CloudFormation provisiona e configura esses recursos para você.

Ao usar CloudFormation, você pode reutilizar seu modelo para configurar seus HealthImaging recursos de forma consistente e repetida. Descreva seus recursos uma vez e, em seguida, provisione os mesmos recursos repetidamente em várias Contas da AWS regiões.

HealthImaging e CloudFormation modelos

Para provisionar e configurar recursos HealthImaging e serviços relacionados, você deve entender CloudFormation os modelos. Os modelos são arquivos de texto formatados em JSON ou YAML. Esses modelos descrevem os recursos que você deseja provisionar em suas CloudFormation pilhas. Se você não estiver familiarizado com JSON ou YAML, você pode usar o CloudFormation Designer para ajudá-lo a começar a usar modelos. CloudFormation Para obter mais informações, consulte O que é o Designer CloudFormation ? no Manual do usuário do AWS CloudFormation .

A AWS HealthImaging oferece suporte à criação de armazenamentos de dados com CloudFormation. Para obter mais informações, incluindo exemplos de modelos JSON e YAML para provisionamento de HealthImaging datastores, consulte a referência do tipo de HealthImaging recurso da AWS no Guia do usuário.AWS CloudFormation

Saiba mais sobre CloudFormation

Para saber mais sobre isso CloudFormation, consulte os seguintes recursos:

- AWS CloudFormation
- AWS CloudFormation Guia do usuário
- Referência de API do CloudFormation
- AWS CloudFormation Guia do usuário da interface de linha de comando

AWS HealthImaging e endpoints VPC de interface ()AWS PrivateLink

Você pode estabelecer uma conexão privada entre sua VPC e criar uma AWS HealthImaging interface VPC endpoint. Os endpoints de interface são alimentados por AWS PrivateLinkuma tecnologia que você pode usar para acessar de forma privada HealthImaging APIs sem um gateway de internet, dispositivo NAT, conexão VPN ou conexão do AWS Direct Connect. As instâncias em sua VPC não precisam de endereços IP públicos para se comunicar. HealthImaging APIs O tráfego entre sua VPC e HealthImaging o tráfego não sai da rede Amazon.

Cada endpoint de interface é representado por uma ou mais Interfaces de Rede Elástica nas sub-redes.

Para obter mais informações, consulte Interface VPC endpoints (AWS PrivateLink) no Guia do usuário da Amazon VPC.

Tópicos

- Considerações sobre HealthImaging VPC endpoints
- Criação de uma interface VPC endpoint para HealthImaging
- Criação de uma política de VPC endpoint para HealthImaging

Considerações sobre HealthImaging VPC endpoints

Antes de configurar uma interface para o VPC endpoint HealthImaging, certifique-se de revisar as propriedades e limitações do endpoint da interface no Guia do usuário do Amazon VPC.

HealthImaging suporta fazer chamadas para todas as AWS HealthImaging ações de sua VPC.

Criação de uma interface VPC endpoint para HealthImaging

Você pode criar um VPC endpoint para o HealthImaging serviço usando o console Amazon VPC ou o (). AWS Command Line Interface AWS CLI Para obter mais informações, consulte Criar um endpoint de interface no Guia do usuário da Amazon VPC.

Crie VPC endpoints para HealthImaging usar os seguintes nomes de serviço:

- com.amazonaws. *region*.imagiologia médica
- com.amazonaws. *region*. runtime-medical-imaging
- com.amazonaws. *region*. dicom-medical-imaging

> **Note**
>
> O DNS privado deve estar habilitado para uso PrivateLink.

Você pode fazer solicitações de API para HealthImaging usar seu nome DNS padrão para a região, por exemplo,`medical-imaging.us-east-1.amazonaws.com`.

Para mais informações, consulte Acessar um serviço por um endpoint de interface no Guia do usuário da Amazon VPC.

Criação de uma política de VPC endpoint para HealthImaging

É possível anexar uma política de endpoint ao endpoint da VPC que controla o acesso ao HealthImaging. Essa política especifica as seguintes informações:

- A entidade principal que pode executar ações
- As ações que podem ser executadas
- Os recursos nos quais as ações podem ser executadas

Para mais informações, consulte Controlar o acesso a serviços com VPC endpoints no Guia do usuário da Amazon VPC.

Exemplo: política de VPC endpoint para ações HealthImaging

Veja a seguir um exemplo de uma política de endpoint para HealthImaging. Quando anexada a um endpoint, essa política concede acesso às HealthImaging ações para todos os diretores em todos os recursos.

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\":\"*\"}]}"
```

Importação entre contas para AWS HealthImaging

Com a importação entre contas e regiões, você pode importar dados para seu armazenamento de dados a partir de HealthImaging buckets do Amazon S3 localizados em outras regiões suportadas. Você pode importar dados entre AWS contas, contas de propriedade de outras AWS organizações e de fontes de dados abertas, como Imaging Data Commons (IDC), localizadas no Registry of Open Data em. AWS

HealthImaging Os casos de uso de importação entre contas/regiões incluem:

- Produtos SaaS de imagens médicas importando dados DICOM de contas de clientes
- Grandes organizações preenchendo um armazenamento de HealthImaging dados a partir de vários buckets de entrada do Amazon S3
- Pesquisadores compartilham dados com segurança em estudos clínicos de várias instituições

Para usar a importação entre contas

1. O proprietário do bucket de entrada (origem) do Amazon S3 deve conceder permissões `s3:ListBucket` e `s3:GetObject` permissões ao proprietário do armazenamento de HealthImaging dados.
2. O proprietário do armazenamento de HealthImaging dados deve adicionar o bucket do Amazon S3 ao IAM. `ImportJobDataAccessRole` Consulte Criar um perfil do IAM para importação.
3. O proprietário do armazenamento de HealthImaging dados deve fornecer o `inputOwnerAccountId`para o bucket de entrada do Amazon S3 ao iniciar o trabalho de importação.

> **Note**
>
> Ao fornecer o`inputOwnerAccountId`, o proprietário do armazenamento de dados valida que o bucket Amazon S3 de entrada pertence à conta especificada para manter a conformidade com os padrões do setor e mitigar possíveis riscos de segurança.

O exemplo de `startDICOMImportJob` código a seguir inclui o `inputOwnerAccountId` parâmetro opcional, que pode ser aplicado a todos, AWS CLI e os exemplos de código do SDK na Como iniciar trabalho de trabalho de trabalho de trabalho seção.

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 "";
}
```

Resiliência na AWS HealthImaging

A infraestrutura AWS global é construída em torno Regiões da AWS de zonas de disponibilidade. Regiões da AWS fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Se você precisa replicar seus dados ou aplicações para distâncias geográficas maiores, use as regiões locais da AWS . Uma região AWS local é um único data center projetado para complementar uma AWS região existente. Como todas Regiões da AWS, as regiões AWS locais estão completamente isoladas das outras Regiões da AWS.

Para obter mais informações sobre zonas de disponibilidade Regiões da AWS e zonas de disponibilidade, consulte Infraestrutura AWS global.

Material de HealthImaging referência da AWS

> **Note**
>
> Todas as ações e tipos de dados da HealthImaging API nativa estão descritos na Referência de HealthImaging API da AWS.

Tópicos

- Suporte DICOM para AWS HealthImaging
- HealthImaging Referência da AWS

Suporte DICOM para AWS HealthImaging

A AWS HealthImaging oferece suporte a elementos DICOM específicos e sintaxes de transferência. Familiarize-se com os elementos de dados DICOM compatíveis em nível de paciente, estudo e série, pois as chaves de HealthImaging metadados são baseadas neles. Antes de iniciar uma importação, verifique se seus dados de imagens médicas estão em conformidade com as sintaxes HealthImaging de transferência suportadas e com as restrições de elementos DICOM.

> **Note**
>
> Atualmente, HealthImaging a AWS não oferece suporte a imagens de segmentação binária ou dados de pixels de sequência de imagens de ícones.

Tópicos

- Classes de SOP compatíveis
- Normalização de metadados
- Sintaxes de transferência compatíveis
- Restrições de elementos de DICOM
- Restrições de metadados de DICOM

Classes de SOP compatíveis

Com a AWS HealthImaging, você pode importar instâncias DICOM P10 Service-Object Pair (SOP) codificadas com qualquer UID de classe SOP, inclusive descontinuada e privada. Todos os atributos privados também são preservados.

Normalização de metadados

Quando você importa seus dados DICOM P10 para a AWS HealthImaging, eles são transformados em conjuntos de imagens compostos por metadados e quadros de imagem (dados em pixels). Durante o processo de transformação, as chaves de HealthImaging metadados são geradas com base em uma versão específica do padrão DICOM. HealthImaging atualmente gera e suporta chaves de metadados com base no Dicionário de Dados DICOM PS3 6.6 2022b.

A AWS HealthImaging oferece suporte aos seguintes elementos de dados DICOM nos níveis de paciente, estudo e série.

Elementos de nível de pacientes

> **Note**
>
> Para obter uma descrição detalhada de cada elemento no nível do paciente, consulte o Registro de elementos de dados DICOM.

A AWS HealthImaging oferece suporte aos seguintes elementos em nível de paciente:

```
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
```

Elementos do nível de estudo

> **Note**
>
> Para obter uma descrição detalhada de cada elemento no nível do estudo, consulte o Registro de elementos de dados DICOM.

A AWS HealthImaging oferece suporte aos seguintes elementos de nível de estudo:

```
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
```

Elementos de nível de séries

> **Note**
>
> Para obter uma descrição detalhada de cada elemento no nível de série, consulte o Registro de elementos de dados DICOM.

A AWS HealthImaging oferece suporte aos seguintes elementos de nível de série:

```
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
```

Sintaxes de transferência compatíveis

A AWS HealthImaging oferece suporte à importação de arquivos DICOM P10 com diferentes sintaxes de transferência. Alguns arquivos mantêm a codificação da sintaxe de transferência original durante a importação, enquanto a maioria dos quadros de imagem sem perdas são transcodificados durante a importação. O comportamento da transcodificação depende da configuração do seu armazenamento de dados. Os armazenamentos de dados usam HTJ2 K como formato de armazenamento por padrão, mas podem ser configurados no momento da criação para usar o JPEG 2000 Lossless. O exemplo a seguir mostra como HealthImaging registra um `StoredTransferSyntaxUID` para cada instância nos metadados retornados por `GetImageSetMetadata`.

```
"Instances": {
   "999.999.2.19941105.134500.2.101": {
       "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.90",
       "ImageFrames": [{ ...
```

> **Note**
>
> Lembre-se desses pontos ao visualizar a tabela a seguir:
>
> - Uma entrada UID de sintaxe de transferência marcada com um asterisco (*) indica que um arquivo está armazenado em seu formato codificado original durante a importação. Para esses arquivos, os metadados `StoredTransferSyntaxUID` localizados na instância correspondem à sintaxe de transferência original.
> - Uma entrada UID da sintaxe de transferência marcada sem um asterisco indica que um arquivo foi transcodificado para HTJ2 K sem perdas durante a importação e armazenado em. HealthImaging O `StoredTransferSyntaxUID` elemento dos metadados da instância será definido no formato de armazenamento do JPEG 2000 de alto rendimento com opções RPCL Image Compression — Lossless Only (1.2.840.10008.1.2.4.202).

- Se a `StoredTransferSyntaxUID` chave não existir ou estiver definida como`null`, você pode presumir que ela está codificada no formato de armazenamento de dados configurado.

HealthImaging sintaxes de transferência suportadas

Sintaxe de transferência UID	Nome da sintaxe de transferência
1.2.840.10008.1.2	Implicit VR Endian: sintaxe de transferência padrão para DICOM
1.2.840.10008.1.2.1* (a segmentação binária retém a codificação original, enquanto a segmentação não binária é transcodificada para K Lossless RPCL) HTJ2	VR explícita em Little Endian
1.2.840.10008.1.2.1.99	VR explícita esvaziada em Little Endian
1.2.840.10008.1.2.2	VR explícita em Big Endian
1.2.840.10008.1.2.4.50*	Linha de base JPEG (Processo 1): Sintaxe de transferência padrão para compactação de imagem JPEG com perdas de 8 bits
1.2.840.10008.1.2.4.51	Linha de base JPEG (Processos 2 e 4): Sintaxe de transferência padrão para compactação de imagem JPEG de 12-bit com perdas (apenas Processo 4)
1.2.840.10008.1.2.4.57	JPEG sem perdas não hierárquico (Processo 14)
1.2.840.10008.1.2.4.70	Previsão de JPEG sem perdas, não hierárqui ca e de primeira ordem (processos 14 [Valor de seleção 1]): sintaxe de transferência padrão para compactação de imagem JPEG sem perdas

Sintaxe de transferência UID	Nome da sintaxe de transferência
1.2.840.10008.1.2.4.80	Compressão de imagem sem perdas JPEG-LS
1.2.840.10008.1.2.4.81	Compressão de imagem JPEG-LS com perdas (quase sem perdas)
1.2.840.10008.1.2.4.90	Compressão de imagem sem perdas JPEG 2000 (apenas sem perda)
1.2.840.10008.1.2.4.91*	Compressão de imagem JPEG 2000
1.2.840.10008.1.2.4.92	Compressão de imagem multicomponente JPEG 2000 Parte 2 (somente sem perdas)
1.2.840.10008.1.2.4.93	Compressão de imagem multicomponente JPEG 2000 Parte 2
1.2.840.10008.1.2.4.112*	JPEG XL
1.2.840.10008.1.2.4.201	Compressão de imagem JPEG 2000 de alto rendimento (somente sem perdas)
1.2.840.10008.1.2.4.202	JPEG 2000 de alto rendimento com opções de compressão de imagem RPCL (somente sem perdas)
1.2.840.10008.1.2.4.203*	Compressão de imagem JPEG 2000 de alto rendimento
1.2.840.10008.1.2.5	RLE sem perdas
1.2.840.10008.1.2.4.100*, 1.2.840.10008.1.2.4.100.1*	MPEG2 Perfil principal Nível principal
1.2.840.10008.1.2.4.101*, 1.2.840.10008.1.2.4.101.1*	MPEG2 Perfil principal em alto nível
1.2.840.10008.1.2.4.102*, 1.2.840.10008.1.2.4.102.1*	MPEG-4 AVC/H.264 Alto Perfil/Nível 4.1

Sintaxe de transferência UID	Nome da sintaxe de transferência
1.2.840.10008.1.2.4.103*, 1.2.840.10008.1.2.4.103.1*	Alto perfil /nível 4.1 compatível com MPEG-4 AVC/H.264 BD
1.2.840.10008.1.2.4.104*, 1.2.840.10008.1.2.4.104.1*	MPEG-4 AVC/H.264 de alto perfil/nível 4.2 para vídeo 2D
1.2.840.10008.1.2.4.105*, 1.2.840.10008.1.2.4.105.1*	MPEG-4 AVC/H.264 de alto perfil/Nível 4.2 do vídeo ITU-T H.264
1.2.840.10008.1.2.4.106*, 1.2.840.10008.1.2.4.106.1*	Alto perfil estéreo MPEG-4 AVC/H.264/Nível 4.2 do vídeo ITU-T H.264
1.2.840.10008.1.2.4.107*	Perfil principal HEVC/H.265/Vídeo de nível 5.1
1.2.840.10008.1.2.4.108*	Perfil HEVC/H.265 Principal 10 /Nível 5.1

*Mantém a codificação da sintaxe de transferência original durante a importação

Restrições de elementos de DICOM

Ao importar seus dados de imagens médicas para a AWS HealthImaging, restrições de comprimento máximo são aplicadas aos seguintes elementos DICOM. Para obter uma importação bem-sucedida, certifique-se de que seus dados não excedam as restrições de tamanho máximo.

Restrições do elemento DICOM durante a importação

Palavra-chave DICOM	Etiqueta DICOM	Comprimento máximo
PatientName	(0010,0010)	256
ID do paciente	(0010,0020)	256
PatientBirthDate	(0010,0030)	18
PatientSex	(0010,0040)	16
StudyInstanceUID	(0020,000D)	256

Palavra-chave DICOM	Etiqueta DICOM	Comprimento máximo
ID do estudo	(0020,0010)	256
StudyDescription	(008,1030)	256
NumberOfStudyRelatedSeries	(0020,1206)	1.000.000
NumberOfStudyRelatedInstances	(0020,1208)	1.000.000
AccessionNumber	(0008,0050)	256
StudyDate	(0008,0020)	18
StudyTime	(0008,0030)	28
SOPInstanceUID	(0008.0018)	256
SeriesInstanceUID	(0020.000E)	256

Restrições de metadados de DICOM

Quando você usa `UpdateImageSetMetadata` para atualizar atributos de HealthImaging metadados, as seguintes restrições DICOM são aplicadas.

- Não é possível atualizar ou remover atributos privados em atributos de Patient/Study/Series/Instance nível, a menos que a restrição de atualização se aplique a ambos e `updatableAttributes` `removableAttributes`
- Não é possível atualizar os seguintes atributos HealthImaging gerados pela AWS: `SchemaVersion` `DatastoreID` `ImageSetIDPixelData`,`Checksum`,,`Width`,`Height`,`MinPixelValue`,`MaxPixelValue`, `FrameSizeInBytes`
- Não é possível atualizar os seguintes atributos DICOM, a menos que o `force` sinalizador esteja definido:`Tag.PixelData`,`Tag.StudyInstanceUID`,,`Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID` `Tag.StudyID`
- Não é possível atualizar atributos com o tipo VR SQ (atributos aninhados), a menos que o `force` sinalizador esteja definido

- Não é possível atualizar atributos de vários valores, a menos que o `force` sinalizador esteja definido
- Não é possível atualizar atributos com valores que não sejam compatíveis com o tipo de atributo VR, a menos que o `force` sinalizador esteja definido
- Não é possível atualizar atributos que não são considerados atributos válidos de acordo com o padrão DICOM, a menos que o `force` sinalizador esteja definido
- Não é possível atualizar atributos entre módulos. Por exemplo, se um atributo de nível de paciente for fornecido no nível de estudo na solicitação de carga útil do cliente, a solicitação poderá ser invalidada.
- Não é possível atualizar atributos se o módulo de atributo associado não estiver presente no `ImageSetMetadata` existente. Por exemplo, você não tem permissão para atualizar atributos para o `seriesInstanceUID` se a Série com `seriesInstanceUID` não estiver presente nos metadados do conjunto de imagens existente.

HealthImaging Referência da AWS

Esta seção contém dados de apoio que são relevantes para a AWS HealthImaging.

Tópicos

- HealthImaging Endpoints e cotas da AWS
- Limites de HealthImaging limitação da AWS
- Verificação de dados de HealthImaging pixels da AWS
- HealthImaging Códigos de aviso
- Bibliotecas de decodificação de quadros de imagem para AWS HealthImaging
- HealthImaging Exemplos de projetos da AWS
- Usando esse serviço com um AWS SDK

HealthImaging Endpoints e cotas da AWS

Os tópicos a seguir contêm informações sobre endpoints e HealthImaging cotas de serviços da AWS.

Tópicos

- Service endpoints
- Cotas de serviço

Service endpoints

Um endpoint de serviço é um URL que identifica um host e a porta como o ponto de entrada para um serviço web. Cada solicitação de serviço da web contém um endpoint. A maioria dos AWS serviços fornece endpoints para regiões específicas para permitir uma conectividade mais rápida. A tabela a seguir lista os endpoints de serviço da AWS HealthImaging.

Nome da região	Região	Endpoint	Protocolo
Leste dos EUA (Norte da Virgínia)	us-east-1	medical-imaging.us-east-1.amazonaws.com medical-imaging-fips.us-east-1.amazonaws.com	HTTPS HTTPS
Oeste dos EUA (Oregon)	us-west-2	medical-imaging.us-west-2.amazonaws.com medical-imaging-fips.us-west-2.amazonaws.com	HTTPS HTTPS
Ásia-Pacífico (Sydney)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS
Europa (Irlanda)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS

Se você estiver usando solicitações HTTP para chamar HealthImaging ações da AWS, deverá usar endpoints diferentes, dependendo das ações que estão sendo chamadas. O menu a seguir lista os endpoints de serviço disponíveis para solicitações HTTP e as ações compatíveis.

Ações de API compatíveis para solicitações HTTP

data store, import, tagging

As seguintes ações de armazenamento, importação e marcação de dados podem ser acessadas por meio do endpoint:

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- Start DICOMImport Job
- Consiga DICOMImport um emprego
- Listar DICOMImport trabalhos
- TagResource
- ListTagsForResource
- UntagResource

image set

As seguintes ações do conjunto de imagens podem ser acessadas por meio do endpoint:

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame

- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging oferece representações dos serviços do DICOMweb Retrieve WADO-RS. Para obter mais informações, consulte Recuperando dados DICOM de HealthImaging.

Os seguintes DICOMweb serviços podem ser acessados via endpoint:

```
https://dicom-medical-imaging.region.amazonaws.com
```

- `GetDICOMInstance`
- `GetDICOMInstanceMetadata`
- `GetDICOMInstanceFrames`

Cotas de serviço

As cotas de serviço são definidas como o valor máximo para seus recursos, ações e itens em sua AWS conta.

> **Note**
>
> É possível solicitar aumentos de cota para cotas ajustáveis do IAM usando o console do Service Quotas. Para obter mais informações, consulte Solicitando um Aumento de Cota no Guia do Usuário do Service Quotas.

A tabela a seguir lista as cotas padrão para a AWS HealthImaging.

Nome	Padrão	Ajustá	Description
Máximo de CopyImageSet solicitações simultâneas por armazenamento de dados	Cada região compatível: 100	Sim	O máximo de CopyImage Set solicitações simultâne as por armazenamento de dados na região atual AWS
Máximo de DeleteImageSet solicitaç ões simultâneas por armazenamento de dados	Cada região compatível: 100	Sim	O máximo de DeleteIma geSet solicitações simultâneas por armazenamento de dados na região atual AWS
Máximo de UpdateImageSetMeta data solicitações simultâneas por armazenamento de dados	Cada região compatível: 100	Sim	O máximo de UpdateIma geSetMetadata solicitaç ões simultâneas por armazenamento de dados na região atual AWS
Máximo de trabalhos de importação simultâneos por datastore	ap-southeast-2: 20 Cada uma das outras regiões compatíveis: 100	Sim	O número máximo de trabalhos de importaçã o simultâneos por armazenamento de dados na região atual AWS
Máximo de datastore	Cada região com suporte: 10	Sim	O número máximo de armazenamentos de dados ativos na AWS região atual
Número máximo de cópias ImageFram es permitidas por solicitação CopyImageSet	Cada região com suporte: 1.000	Sim	O número máximo de cópias ImageFram es permitidas por

Nome	Padrão	Ajustá	Description
			CopyImageSet solicitação na região atual AWS
Número máximo de arquivos em um trabalho de importação DICOM	Cada região compatível: 5.000	Sim	O número máximo de arquivos em uma tarefa de importação DICOM na região atual AWS
Número máximo de pastas aninhadas em um trabalho de importação DICOM	Cada região compatível: 10.000	Não	O número máximo de pastas aninhadas em uma tarefa de importaçã o DICOM na região atual AWS
Limite máximo de tamanho da carga útil (em KB) aceito por UpdateIma geSetMetadata	Cada região compatível: 10 kilobytes	Sim	O limite máximo de tamanho da carga útil (em KB) aceito pela UpdateImageSetMeta data região atual AWS
Tamanho máximo (em GB) de todos os arquivos em um trabalho de importação DICOM	Cada região compatível: 10 gigabytes	Não	O tamanho máximo (em GB) de todos os arquivos em uma tarefa de importação DICOM na região atual AWS
Tamanho máximo (em GB) de cada arquivo DICOM P10 em um trabalho de importação DICOM	Cada região compatível: 4 gigabytes	Não	O tamanho máximo (em GB) de cada arquivo DICOM P10 na tarefa de importação DICOM na região atual AWS

Nome	Padrão	Ajustá	Description
Limite máximo de tamanho (em MB) ImageSetMetadata por importação, cópia e UpdateImageSet	Cada região compatível: 50 megabytes	Sim	O limite máximo de tamanho (em MB) ImageSetMetadata por importação, cópia e UpdateImageSet na AWS região atual

Limites de HealthImaging limitação da AWS

Sua AWS conta tem limites de limitação que se aplicam às ações de HealthImaging API da AWS. Para todas as ações, um erro `ThrottlingException` é gerado se os limites de controle de utilização forem excedidos. Para obter mais informações, consulte a AWS HealthImaging API Reference.

> **Note**
>
> Os limites de limitação são ajustáveis para todas as ações HealthImaging da API. Para solicitar um ajuste do limite de controle de utilização, entre em contato com o AWS Support Center. Para criar um caso, faça login na sua AWS conta e escolha Criar caso.

A tabela a seguir lista os limites de limitação para HealthImaging ações nativas e representações de DICOMweb serviços.

Limites de HealthImaging limitação da AWS

Ação	Taxa de controle	Explosão do controle
CreateDatastore	0,085 tps	1 tps
GetDatastore	10 tps	20 tps
ListDatastores	5 tps	10 tps
DeleteDatastore	0,085 tps	1 tps
Start DICOMImport Job	1 tps	2 colheres de chá

Ação	Taxa de controle	Explosão do controle
Consiga DICOMImport um emprego	25 tps	50 tps
Listar DICOMImport trabalhos	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
DeleteImageSet	0,25 tps	1 tps
TagResource	10 tps	20 tps
ListTagsForResource	10 tps	20 tps
UntagResource	10 tps	20 tps
DICOMInstanceObtenha*	50 tps	100 tps
Obtenha DICOMInstance metadados*	50 tps	100 tps
Obtenha DICOMInstance quadros*	50 tps	100 tps
Obter DICOMSeries metadados	50 tps	100 tps

*Representação de um serviço DICOMweb

Verificação de dados de HealthImaging pixels da AWS

Durante a importação, HealthImaging fornece verificação de dados de pixel integrada, verificando o estado de codificação e decodificação sem perdas de cada imagem. Esse recurso garante que as imagens decodificadas usando bibliotecas de decodificação HTJ2 K sempre correspondam às imagens DICOM P10 originais importadas. HealthImaging

- O processo de integração de imagens começa quando uma tarefa de importação captura o estado original da qualidade de pixels das imagens DICOM P10 antes de serem importadas. Uma soma de verificação de resolução de quadro de imagem (IFRC) imutável exclusiva é gerada para cada imagem usando o algoritmo. CRC32 O valor da soma de verificação do IFRC é apresentado no documento de metadados`job-output-manifest.json`. Para obter mais informações, consulte Noções básicas sobre as tarefas de importação.
- Depois que as imagens são importadas para um armazenamento de HealthImaging dados e transformadas em conjuntos de imagens, os quadros de imagem HTJ2 codificados em K são imediatamente decodificados e os novos IFRCs são calculados. HealthImaging em seguida, compara a resolução total IFRCs das imagens originais com a nova moldura IFRCs de imagem importada para verificar a precisão.
- Uma condição de erro descritiva correspondente por imagem é capturada no registro de saída do trabalho de importação (`job-output-manifest.json`) para você revisar e verificar.

Para verificar os dados de pixel

1. Depois de importar seus dados de imagens médicas, visualize o sucesso descritivo (ou condição de erro) do conjunto por imagem capturado no registro em log de saída do trabalho de importação, `job-output-manifest.json`. Para obter mais informações, consulte Noções básicas sobre as tarefas de importação.
2. Os conjuntos de imagens são compostos por metadados e quadros de imagem (dados de pixels). Os metadados do conjunto de imagens contêm informações sobre os quadros de imagem associados. Use a `GetImageSetMetadata` ação para obter metadados para um conjunto de imagens. Para obter mais informações, consulte Obtendo metadados do conjunto de imagens.
3. O `PixelDataChecksumFromBaseToFullResolution` contém o IFRC (soma de verificação) para a imagem em resolução total. Para imagens armazenadas na sintaxe de transferência original 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 e 1.2.840.10008.1.2.1 (somente segmentação binária), a soma de verificação é calculada na

imagem original. Para imagens armazenadas em HTJ2 K Lossless com RPCL, a soma de verificação é calculada na imagem de resolução total decodificada. Para obter mais informações, consulte Sintaxes de transferência compatíveis.

Veja a seguir um exemplo de saída de metadados para o IFRC que é gerada como parte do processo de trabalho de importação e registrada em. `job-output-manifest.json`

```
"ImageFrames": [{
"ID": "678906789067890123451234512345 12",
"PixelDataChecksumFromBaseToFullResolution": [
{
    "Width": 512,
    "Height": 512,
    "Checksum": 2510355201
}
]
```

Para imagens armazenadas em sua sintaxe de transferência original 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 e 1.2.840.10008.1.2.1 (somente segmentação binária), o e não estará disponível. `MinPixelValue` `MaxPixelValue` `FrameSizeInBytes`Isso indica o tamanho da moldura original.

```
"PixelDataChecksumFromBaseToFullResolution": [
   {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": null,
"MaxPixelValue": null,
"FrameSizeInBytes": 429
```

Para imagens armazenadas em HTJ2 K Lossless com RPCL, `FrameSizeInBytes` indica o tamanho do quadro de imagem decodificado.

```
"PixelDataChecksumFromBaseToFullResolution": [
   {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": 11,
"MaxPixelValue": 11,
"FrameSizeInBytes": 1652
```

4. Para instâncias que contêm vídeo, HealthImaging executa uma validação leve do codec que verifica se a sintaxe de transferência especificada nos metadados DICOM corresponde ao codec de vídeo.

   HealthImaging calcula um valor de soma de verificação IFRC no objeto de vídeo original usando o algoritmo. CRC32 O valor da soma de verificação do IFRC é registrado no `job-output-manifest.json` e persiste nos metadados. HealthImaging Assim como acontece com as imagens armazenadas em sua sintaxe de transferência original (descrita acima), o `MinPixelValue` e não `MaxPixelValue` estará disponível. FrameSizeInBytes Isso indica o tamanho da moldura original.
5. HealthImaging verifique os dados de pixels, acesse o procedimento de verificação de dados de pixels em GitHub e siga as instruções no `README.md` arquivo para verificar de forma independente o processamento de imagens sem perdas pelas várias Bibliotecas de decodificação de quadros de imagem que são utilizadas pelo. HealthImaging Depois que a imagem completa for carregada, você poderá calcular o IFRC para dados de entrada brutos do seu lado e compará-lo com o valor do IFRC fornecido nos HealthImaging metadados para verificar os dados de pixels.

HealthImaging Códigos de aviso

HealthImaging tenta importar todos os seus dados de imagens médicas. Se não conformidades de dados ou elementos de dados não reconhecidos forem encontradas durante as importações, HealthImaging adicionará um dos seguintes avisos ao arquivo. `warning.ndjson` Um aviso associado a uma instância importada também poderá ser pesquisado por meio da `SearchDICOMInstances` ação com o `WarningReason` elemento. As instâncias importadas com um aviso podem ter reduzido o suporte via HealthImaging APIs, conforme descrito abaixo.

HealthImaging Importar códigos de aviso

Motivo do aviso (hexadecimal)	Motivo do aviso (decimal)	Tipo de aviso (Enum)	Detalhes do aviso	Comportamento resultante
Razões de aviso padrão DICOM				
0xB000	45056	COERÇÃO DE	A ingestão modificou um ou mais elementos	n/a

Motivo do aviso (hexadecimal)	Motivo do aviso (decimal)	Tipo de aviso (Enum)	Detalhes do aviso	Comportamento resultante
		ELEMENTOS DE DADOS	de dados durante o armazenamento da instância. Consulte a Seção 6.6.1.3.	
0xB006	45062	ELEMENTOS_DESCARTADOS	A ingestão descartou alguns elementos de dados durante o armazenamento da instância. Consulte a Seção 6.6.1.3.	n/a
0xB007	45063	SOP_CLASS_DATA_MISMATCH	A `StoreDICOM` ação observou que o conjunto de dados não correspondia às restrições da classe SOP durante o armazenamento da instância.	n/a
		Motivos de HealthImaging aviso da AWS		

Motivo do aviso (hexadecimal)	Motivo do aviso (decimal)	Tipo de aviso (Enum)	Detalhes do aviso	Comportamento resultante
0xB100	45312	EXCEÇÃO_DE_TRANSCODIFICAÇÃO	Esse aviso ocorre quando HealthImaging não é possível transcodificar a instância PixelData para HTJ2 K (formato de armazenamento padrão) ou se ela PixelData não pode ser transcodificada por outro motivo (ou seja, falhas nas validações, atributos de dados de pixel incorretos etc.). Nesse caso, os dados de pixels serão armazenados como um blob.	Os dados de pixel ainda podem ser recuperados como um único blob, se necessário. Se necessário, passar o caractere curinga “*” como cabeçalho `transfer-syntax` em Aceitar os retornará no formato armazenado.
0xB110	45328	FALHA NA EXTRAÇÃO DE QUADROS	Esse aviso ocorre quando há um problema ao analisar quadros individuais PixelData com base nos metadados DICOM fornecidos.	Os dados de pixel foram malformados e não podem ser recuperados, use `GetDICOMInstance` para recuperar a instância inteira
0xB111	45329	INCOMPATIBILIDADE DE NÚMERO_DE_QUADRO	Esse aviso ocorre quando o elemento DICOM NumberOfFrames "" não corresponde ao número real de “fragmentos” da imagem no arquivo DICOM de entrada.	Os dados de pixel foram malformados e não podem ser recuperados, use `GetDICOMInstance` para recuperar a instância inteira

Motivo do aviso (hexadecimal)	Motivo do aviso (decimal)	Tipo de aviso (Enum)	Detalhes do aviso	Comportamento resultante
0xB112	45330	TABELA_OFFSET_INVÁLIDA	Esse aviso ocorre quando a tabela de deslocamento nos fragmentos do arquivo DICOM de entrada não corresponde ao tamanho real do quadro e pode resultar em quadros malformados, dependendo da gravidade.	Os dados de pixel foram malformados e não podem ser recuperados, use `GetDICOMInstance` para recuperar a instância inteira
0xB120	45344	SINTAXE_TRANSFERÊNCIA_NÃO SUPORTADA	Esse aviso ocorre quando HealthImaging encontra uma sintaxe de transferência não reconhecida ou não suportada. Quando isso acontecer, HealthImaging armazenará os dados de pixels como uma bolha.	Os dados de pixel ainda podem ser recuperados como um único blob, se necessário. Se necessário, passar o caractere curinga “*” como cabeçalho `transfer-syntax` em Aceitar os retornará no formato armazenado.
0xB201	45570	FORMATO_UID INVÁLIDO	Esse aviso ocorre quando um ou mais elementos UID violam a representação de valor DICOM (por exemplo) `1.2.3..4`	n/a

Motivo do aviso (hexadecimal)	Motivo do aviso (decimal)	Tipo de aviso (Enum)	Detalhes do aviso	Comportamento resultante
0xB202	45571	INVALID_DICOM_VALUE_LENGTH	Esse aviso ocorre quando um elemento DICOM tem um comprimento maior do que o suportado pela representação de valor DICOM, potencialmente introduzindo comportamentos inválidos para ações. search/retrieve	Alguns campos podem não ser analisáveis e, portanto, não podem ser pesquisados (ou seja, ou`StudyDate`) `StudyTime`
0xBFFF	47513	OUTRO	Esse aviso ocorre quando há um aviso não detectado que HealthImaging não é capturado como um código de aviso específico.	Os dados de pixel foram malformados e não podem ser recuperados, use `GetDICOMInstance` para recuperar a instância inteira

Bibliotecas de decodificação de quadros de imagem para AWS HealthImaging

Durante a importação, algumas sintaxes de transferência mantêm sua codificação original, enquanto outras são transcodificadas para High-Throughput JPEG 2000 (HTJ2K) sem perdas por padrão ou JPEG 2000 Lossless (quando configuradas), dependendo da configuração do seu armazenamento de dados. HTJ2K oferece exibição de imagens consistentemente rápida e acesso universal aos recursos avançados do HTJ2 K. Como os quadros de imagem são codificados em HTJ2 K ou JPEG 2000 Lossless durante a importação, eles devem ser decodificados antes da visualização em um visualizador de imagens. Para obter informações sobre como determinar as sintaxes de transferência, consulte Sintaxes de transferência suportadas. Para obter informações sobre como determinar as sintaxes de transferência, consulteSintaxes de transferência compatíveis.

> **Note**
>
> HTJ2K é definido na Parte 15 da norma JPEG2 000 (ISO/IEC 15444-15:2019). HTJ2K mantém os recursos avançados do JPEG2 000, como escalabilidade de resolução, recintos, ladrilhos, alta profundidade de bits, vários canais e suporte ao espaço de cores.

Tópicos

- Bibliotecas de decodificação de quadros de imagem
- Visualizadores de imagens

Bibliotecas de decodificação de quadros de imagem

Dependendo da sua linguagem de programação, recomendamos as seguintes bibliotecas de decodificação para decodificar quadros de imagem.

- NVIDIA nv JPEG2 000 — comercial, acelerado por GPU
- Kakadu Software: comercial, C++ com vínculos Java e .NET
- OpenJPH: código aberto, C++ e WASM
- OpenJPEG: código aberto, C/C++, Java
- openjphpy código aberto, Python
- pylibjpeg-openjpeg: código aberto, Python

Visualizadores de imagens

Você pode ver os quadros de imagem depois de decodificá-los. As ações de HealthImaging API da AWS oferecem suporte a uma variedade de visualizadores de imagens de código aberto, incluindo:

- Open Health Imaging Foundation (OHIF)
- Cornerstone.js

HealthImaging Exemplos de projetos da AWS

HealthImaging A AWS fornece os seguintes exemplos de projetos em GitHub.

Visualizador OHIF integrado à AWS HealthImaging via OIDC

Este AWS Cloud Development Kit (AWS CDK)projeto implanta o visualizador OHIF na Amazon. CloudFront O visualizador é integrado a um armazenamento de dados da Amazon Web Services como fonte de DICOMWeb dados e com o Amazon Cognito como provedor de identidade para autenticação via OIDC.

Ingestão de DICOM do local para a AWS HealthImaging

Um projeto AWS sem servidor para implantar uma solução de ponta de IoT que recebe arquivos DICOM de uma fonte DICOM DIMSE (PACS, VNA, tomógrafo) e os armazena em um bucket seguro do Amazon S3. A solução indexa os arquivos DICOM em um banco de dados e enfileira cada série DICOM para ser importada na AWS. HealthImaging É composto por um componente executado na borda que é gerenciado por AWS IoT Greengrass, e um pipeline de ingestão de DICOM executado na nuvem. AWS

Proxy Tile Level Marker (TLM)

Um AWS Cloud Development Kit (AWS CDK)projeto para recuperar quadros de imagem da AWS HealthImaging usando marcadores de nível de bloco (TLM), um recurso do High-Throughput JPEG 2000 (K). HTJ2 Isso resulta em tempos de recuperação mais rápidos com imagens de baixa resolução. Os possíveis fluxos de trabalho incluem a geração de miniaturas e o carregamento progressivo de imagens.

CloudFront Entrega na Amazon

Um projeto AWS sem servidor para criar uma CloudFront distribuição da Amazon com um endpoint HTTPS que armazena em cache (usando GET) e entrega quadros de imagem a partir da borda. Por padrão, o endpoint autentica solicitações com um token web JSON (JWT) do Amazon Cognito. Tanto a autenticação quanto a assinatura da solicitação são feitas na borda usando o Lambda @Edge. Esse serviço é um recurso da Amazon CloudFront que permite que você execute o código mais perto dos usuários do seu aplicativo, o que melhora o desempenho e reduz a latência. Não há infraestrutura para gerenciar.

Interface do usuário do AWS HealthImaging Viewer

Um AWS Amplifyprojeto para implantar uma interface de usuário de front-end com autenticação de back-end com a qual você pode visualizar atributos de metadados do conjunto de imagens e quadros de imagem (dados de pixels) armazenados na AWS HealthImaging usando decodificação progressiva. Opcionalmente, você pode integrar os projetos Tile Level Marker (TLM) Proxy and/or Amazon CloudFront Delivery acima para carregar quadros de imagem usando um método alternativo.

HealthImaging DICOMweb Proxy AWS

Um projeto baseado em Python para habilitar terminais DICOMweb WADO-RS e QIDO-RS em um armazenamento de HealthImaging dados para oferecer suporte a visualizadores de imagens médicas baseados na web e outros aplicativos compatíveis. DICOMweb

> **Note**
>
> Este projeto não usa HealthImaging a representação DICOMweb APIs descrita emUsando DICOMweb com a AWS HealthImaging.

Para ver exemplos adicionais de projetos, consulte AWS HealthImaging Samples on GitHub.

Usando esse serviço com um AWS SDK

AWS kits de desenvolvimento de software (SDKs) estão disponíveis para muitas linguagens de programação populares. Cada SDK fornece uma API, exemplos de código e documentação que permitem que os desenvolvedores criem facilmente aplicações em seu idioma de preferência.

Documentação do SDK	Exemplos de código
AWS SDK para C++	AWS SDK para C++ exemplos de código
AWS CLI	AWS CLI exemplos de código
AWS SDK para Go	AWS SDK para Go exemplos de código
AWS SDK para Java	AWS SDK para Java exemplos de código
AWS SDK para JavaScript	AWS SDK para JavaScript exemplos de código
AWS SDK para Kotlin	AWS SDK para Kotlin exemplos de código
AWS SDK para .NET	AWS SDK para .NET exemplos de código
AWS SDK para PHP	AWS SDK para PHP exemplos de código
Ferramentas da AWS para PowerShell	Ferramentas da AWS para PowerShell exemplos de código

Documentação do SDK	Exemplos de código
AWS SDK para Python (Boto3)	AWS SDK para Python (Boto3) exemplos de código
AWS SDK para Ruby	AWS SDK para Ruby exemplos de código
AWS SDK para Rust	AWS SDK para Rust exemplos de código
AWS SDK para SAP ABAP	AWS SDK para SAP ABAP exemplos de código
AWS SDK para Swift	AWS SDK para Swift exemplos de código

Exemplo de disponibilidade

Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na parte inferior desta página.

Otimização de custo

HealthImaging foi projetado para otimizar os custos de armazenamento movendo automaticamente os dados para o nível de acesso mais econômico quando os padrões de acesso mudam. À medida que os padrões de uso mudam com o tempo, a hierarquização inteligente fornece gerenciamento automático do ciclo de vida dos dados DICOM sem nenhuma sobrecarga operacional. HealthImaging tem dois níveis de armazenamento:

- Armazenamento em camadas de acesso frequente para dados DICOM recém-importados ou acessados regularmente.
- Arquive o armazenamento em camadas de acesso instantâneo para dados DICOM que não foram acessados recentemente. Oferece custos reduzidos de armazenamento para arquivamento de longo prazo, mantendo latências de acesso em milissegundos.

Você é cobrado pelo tamanho de armazenamento agregado de todos os conjuntos de imagens em um armazenamento de dados. Ambos os níveis de armazenamento são cobrados por GB por mês, e não há taxa por conjunto de imagens. Também não há cobranças de recuperação para mover dados entre níveis de armazenamento.

> **Note**
>
> Os conjuntos de imagens são cobrados com um tamanho mínimo de 5 MB. HealthImaging aceita conjuntos de imagens menores que 5 MB, mas esses objetos menores são cobrados a uma taxa de 5 MB.

Como funciona a hierarquização inteligente

Seus dados são armazenados na camada de Acesso Frequente quando novos conjuntos de imagens são criados. Os conjuntos de imagens podem ser criados importando novos dados (por meio de Import Jobs ou DICOMweb STOW-RS) ou invocando. CopyImageSet HealthImaging move automaticamente conjuntos de imagens que não foram acessados por 30 dias consecutivos para a camada Archive Instant Access, e os conjuntos de imagens permanecem na camada Archive Instant Access até serem acessados novamente.

A seguir estão as ações que constituem acesso aos dados DICOM que movem automaticamente conjuntos de imagens da camada Archive Instant Access de volta para a camada de Acesso Frequente:

- Invocação GetImageSetMetadataou qualquer ação do DICOMweb WADO-RS. GetImageFrame
- Invocando CopyImageSetou UpdateImageSetMetadata. No caso das operações de cópia, somente os conjuntos de imagens replicados são escalonados para o Acesso Frequente. No caso de Copiar com destino, o conjunto de imagens de destino é dividido em camadas.
- Visualizando e baixando dados do conjunto de imagens por meio do console AWS HealthImaging de gerenciamento.

As ações acima promovem conjuntos de imagens para o nível Acesso Frequente e evitam que os conjuntos de imagens no nível Acesso Frequente sejam reduzidos para o nível Archive Instant Access por mais 30 dias. O acesso aos conjuntos de imagens pode ocorrer por meio do console AWS de gerenciamento ou por meio de interfaces programáticas, como o AWS CLI ou AWS SDKs. Outras ações não constituem acesso e, portanto, não moverão automaticamente objetos do nível Archive Instant Access de volta para o nível de Acesso Frequente. Veja a seguir uma amostra, não uma lista definitiva, destas ações:

- Ações em armazenamentos de dados (CreateDatastoree GetDatastore)
- Listar ações (Listar DICOMImport trabalhos ListImageSetVersions, e ListTagsForResource)
- Ações de pesquisa (SearchImageSetse consultas DICOMweb QIDO-RS)
- Invocando GetImageSetTagResource, ou UntagResource
- Operações de exclusão

Os dados importados para HealthImaging são cobrados por uma duração mínima de armazenamento de 30 dias. Você pode excluir seus dados a qualquer momento, mas cada imagem excluída antes de 30 dias após a importação será cobrada pelo restante da duração mínima.

Estimando o armazenamento de dados estruturados

HealthImaging armazena algumas informações do cabeçalho DICOM no armazenamento indexado. Você é cobrado por esse consumo de armazenamento estruturado, independente do nível de armazenamento do conjunto de imagens, e essas cobranças são aditivas. Você pode estimar o consumo de armazenamento estruturado multiplicando o número de recursos DICOM em seus

armazenamentos de dados pelos tamanhos de registros indexados por recurso. Os valores a seguir são aproximados.

Recurso DICOM	Tamanhos indexados (bytes)
Estude	1024
Série	830
Instância	680

HealthImaging Lançamentos da AWS

A tabela a seguir mostra quando os recursos e atualizações foram lançados para o HealthImaging serviço e a documentação da AWS. Para obter mais informações, consulte o tópico vinculado.

Alteração	Descrição	Data
CloudWatch Métricas da Amazon para HealthImaging	HealthImaging publica métricas adicionais CloudWatch no `AWS/HealthImaging` namespace, incluindo métricas de uso de recursos nos níveis da conta e do armazenamento de dados. Para obter mais informações, consulte Usando a Amazon CloudWatch com HealthImaging.	12 de fevereiro de 2026
Função vinculada ao serviço para HealthImaging	HealthImaging agora oferece suporte a funções vinculadas a serviços que permitem que o serviço publique métricas do armazenamento de dados em seu CloudWatch nome. A `AWSServiceRoleForHealthImaging` função pode ser criada usando o console do IAM ou o AWS CLI. Para obter mais informações, consulte Usando funções vinculadas a serviço para HealthImaging.	9 de fevereiro de 2026
AWSHealthImagingServiceRolePolicy política gerenciada	HealthImaging adicionou uma nova política gerenciada `AWSHealthImagingServiceRolePolicy` para	9 de fevereiro de 2026

	a função vinculada ao serviço que fornece permissões para gerenciar operações de serviço e publicar métricas de serviço em. CloudWatch Para obter mais informaçõ es, consulte AWSHealth ImagingServiceRolePolicy.	
Suporte ao JPEG XL para armazenamentos de dados da AWS HealthImaging	HealthImaging suporta a importação e o armazenam ento de arquivos DICOM com perdas na sintaxe de transferê ncia JPEG XL (1.2.840. 10008.1.2.4.112). Para obter mais informações, consulte Sintaxes de transferência suportadas.	2 de fevereiro de 2026

Importação DICOM aprimorada com suporte de aviso para dados não conformes	HealthImaging agora importa dados DICOM anteriormente rejeitados aceitando arquivos não compatíveis e fornecendo avisos detalhados por meio de eventos. EventBridge Os trabalhos de importação agora geram uma pasta WARNING contendo `warning.ndjson` arquivos com códigos de motivo de aviso específicos para dados não conformes. A atualização adiciona suporte a elementos DICOM pesquisáveis WarningReason (0008.1196) e introduz `transfer-syntax=*` parâmetros para recuperar instâncias em seu formato armazenado quando a transcodificação não é viável. Consulte Entendendo tarefas de importação e códigos de HealthImaging aviso para obter mais informações.	8 de dezembro de 2025

Suporte ao JPEG 2000 Lossless para armazenamentos de dados da AWS HealthImaging	A AWS HealthImaging agora oferece suporte à codificação nativa JPEG 2000 Lossless para imagens médicas, permitindo que você crie datastores que persistam e recuperem quadros de imagem sem perdas no formato JPEG 2000 sem transcodificação. Isso permite uma recuperação de menor latência para aplicativos que exigem o formato JPEG 2000 Lossless quando você especifica `–lossless-storage-format JPEG_2000_LOSSLESS` ao criar um armazenamento de dados.	25 de novembro de 2025

Design de aprimoramentos de pesquisa do QIDO-RS	HealthImaging agora oferece suporte à pesquisa curinga nos principais atributos DICOM (nome do paciente, nome do médico de referência, ID do paciente, descrição do estudo, modalidade e número de acesso) e aos recursos de pesquisa difusa de nomes de Patient/Referring médicos para acomodar termos incompletos ou com erros ortográficos. A atualização também expande os recursos de consulta da API QIDO-RS para incluir pesquisas de data de nascimento do paciente e descrição do estudo, aprimorando a capacidade de localizar estudos e séries relevantes. Consulte Pesquisando dados DICOM HealthImaging para obter mais informações.	15 de setembro de 2025

Autorização do OpenID Connect (OIDC) para DICOMweb APIs	HealthImaging agora suporta autorização de token portador do OpenID Connect (OIDC) em todos. DICOMweb APIs Isso adiciona interoperabilidade OAuth 2.0 baseada em padrões com visualizadores e kits de ferramentas comuns (por exemplo, OHIF, SLIM, MONAI) ao aceitar a emissão de IDP no cabeçalho . JWTs `Authorization` Consulte Autenticação OIDC DICOMweb APIs para obter mais informações.	3 de setembro de 2025
Support para importação DICOMweb de dados e DICOMweb Bulkdata	HealthImaging HealthImaging suporta o armazenamento de arquivos DICOM P10 por meio do protocolo DICOMweb STOW-RS. Consulte Importação de dados para obter mais detalhes. Além disso, HealthImaging suporta DICOM Bulkdata para garantir recuperações de metadados consistentes e de baixa latência. Consulte Obter dados em massa DICOM para obter mais informações.	30 de junho de 2025

Organização automática de dados, pesquisa DICOMweb QIDO-RS e DICOMweb aprimoramentos do WADO-RS	HealthImaging fornece organização automática dos dados DICOM P10 na importação de acordo com a hierarquia de níveis de paciente, estudo e série do padrão DICOM. Para obter mais informações, consulte Noções básicas sobre trabalhos de importação. HealthImaging suporta pesquisa avançada, de acordo com o padrão DICOMweb QIDO-RS. Consulte Pesquisando dados DICOM HealthImaging para obter mais informações. HealthImaging suporta a recuperação dos metadados de todas as instâncias DICOM em uma série por meio de uma única ação de API, conforme descrito em Obter metadados da série DICOM de. HealthImaging	22 de maio de 2025
A criação do conjunto de imagens usa menos elementos DICOM	HealthImaging diminui o número de elementos usados ao agrupar objetos DICOM P10 de entrada em conjuntos de imagens. Para obter mais informações, consulte O que é um conjunto de imagens? .	27 de janeiro de 2025

Suporte com perdas para Start Job DICOMImport	HealthImaging suporta a importação e o armazenamento de arquivos DICOM com perdas (1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50) e arquivos de segmentação binária em seu formato original. Para obter mais informações, consulte Sintaxes de transferência suportadas.	1.º de novembro de 2024
Suporte com perdas para recuperação DICOMweb APIs	HealthImaging suporta a recuperação de imagens e instâncias armazenadas em 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 e 1.2.840.10008.1.2.1 (somente segmentação binária) em seu formato original ou em Explicit VR Little Endian (1.2.840.10008.1.2.1). Para obter mais informações, consulte Sintaxes de transferência suportadas e Recuperação de dados DICOM.	1.º de novembro de 2024
Importações mais rápidas para patologia digital	HealthImaging suporta trabalhos de importação até 6x mais rápidos para patologia digital DICOM (WSI).	1.º de novembro de 2024

Suporte à segmentação binária	HealthImaging suporta a ingestão e a recuperação de arquivos de segmentação binária DICOM. Para obter mais informações, consulte Sintaxes de transferência suportadas.	1.º de novembro de 2024
Reverter para um ID de versão anterior do conjunto de imagens	HealthImaging fornece o `revertToVersionId` parâmetro para reverter para um ID de versão anterior do conjunto de imagens. Para obter mais informaçõ es, consulte `revertToV ersionId` a AWS HealthIma ging API Reference.	24 de julho de 2024

Funcionalidade de força para modificação do conjunto de imagens	HealthImaging fornece o tipo de `Overrides` dados com o parâmetro de `forced` solicitação opcional. A configuração desse parâmetro força as `CopyImageSet` ações `UpdateImageSetMetadata` e, mesmo que os metadados em nível de paciente, estudo ou série sejam incompatíveis. Para obter mais informações, consulte Substituições na referência da HealthImaging API da AWS.
	
	• `UpdateImageSetMetadata` funcionalidade de força — HealthImaging introduz o parâmetro de `force` solicitação opcional para atualizar os seguintes atributos:
	  • `Tag.StudyInstanceUID` , `Tag.SeriesInstanceUID` , `Tag.SOPInstanceUID` , e `Tag.StudyID`
	  • Adicionar, remover ou atualizar elementos de dados DICOM privados no nível da instância
	
	Para obter mais informações, consulte UpdateImageSetMetadataa AWS
		24 de julho de 2024

	HealthImaging API Reference. • `CopyImageSet` funcionalidade de força — HealthImaging introduz o parâmetro de `force` solicitação opcional para copiar conjuntos de imagens. A configuração desse parâmetro força a `CopyImageSet` ação, mesmo que os metadados em nível de paciente, estudo ou série sejam incompatíveis entre e. `sourceImageSet` `destinationImageSet` Nesses casos, os metadados inconsistentes permanecem inalterados no. `destinationImageSet` Para obter mais informações, consulte CopyImageSeta AWS HealthImaging API Reference.	
Copiar subconjuntos de instâncias SOP	HealthImaging aprimora a `CopyImageSet` ação para que você possa escolher uma ou mais instâncias SOP de a `sourceImageSet` para copiar para a. `destinationImageSet` Para obter mais informações, consulte Copiar um conjunto de imagens.	24 de julho de 2024

GetDICOMInstanceMetadata para retornar metadados da instância DICOM	HealthImaging fornece a GetDICOMInstanceMetadata API para retornar metadados (.jsonarquivo) da Parte 10 do DICOM. Para obter mais informações, consulte Como obter metadados da instância.	11 de julho de 2024
GetDICOMInstanceFrames para retornar quadros de instância DICOM (dados em pixels)	HealthImaging fornece a GetDICOMInstanceFrames API para retornar quadros DICOM Parte 10 (multipart solicitação). Para obter mais informações, consulte Como obter quadros de instância.	11 de julho de 2024

Suporte aprimorado para importações de dados DICOM não padrão	HealthImaging fornece suporte para importações de dados que incluem desvios do padrão DICOM. Para obter mais informações, consulte Restrições do elemento DICOM.
	• Os seguintes elementos de dados DICOM podem ter até 256 caracteres de comprimento máximo:
	  • `Patient's Name (0010,0010)`
	  • `Patient ID (0010,0020)`
	  • `Accession Number (0008,0050)`
	• As seguintes variações de sintaxe são permitidas para `Study Instance UIDSeries Instance UID`,`Treatment Session UID`,`Manufacturer’s Device Class UID`,`Device UID`, e`Acquisition UID` :
	  • O primeiro elemento de qualquer UID pode ser zero
	  • UIDs pode começar com um ou mais zeros à esquerda
	  • UIDs pode ter até 256 caracteres
		28 de junho de 2024

Notificações de eventos	HealthImaging se integra à Amazon EventBridge para oferecer suporte a aplicativos orientados por eventos. Para obter mais informações, consulte Usando EventBridge.	5 de junho de 2024
`GetDICOMInstance` para retornar dados da instância DICOM	HealthImaging fornece o `GetDICOMInstance` serviço para retornar dados da instância DICOM Parte 10 (`.dcm`arquivo). Para obter mais informações, consulte Como obter uma instância.	15 de maio de 2024
Importação entre contas	HealthImaging fornece suporte para importações de dados de buckets do Amazon S3 localizados em outras regiões suportadas. Para obter mais informações, consulte Importação entre contas.	15 de maio de 2024

Aprimoramentos de pesquisa para conjuntos de imagens	HealthImaging `SearchImageSets` A ação oferece suporte aos seguintes aprimoramentos de pesquisa. Para obter mais informaçõ es, consulte Pesquisando conjuntos de imagens. • Suporte adicional para pesquisar em `UpdatedAt` e `SeriesInstanceUID` • Pesquise entre a hora de início e a hora de término • Classifique os resultados da pesquisa por `Ascending` ou `Descending` • Os parâmetros da série DICOM são retornados nas respostas	3 de abril de 2024
Aumento do tamanho máximo do arquivo para importações	HealthImaging suporta um tamanho máximo de arquivo de 4 GB para cada arquivo DICOM P10 em uma tarefa de importação. Para obter mais informações, consulte Cotas de serviço.	6 de março de 2024

Sintaxes de transferência para JPEG Lossless e K HTJ2	HealthImaging fornece suporte para as seguintes sintaxes de transferência para importaçõ es de tarefas. Para obter mais informações, consulte Sintaxes de transferência suportadas. • 1.2.840.10008.1.2.4.57 — JPEG sem perdas não hierárquico (Processo 14) • 1.2.840.10008.1.2.4.201 — Compressão de imagem JPEG 2000 de alto rendimento (somente sem perdas) • 1.2.840.10008.1.2.4.202 — JPEG 2000 de alto rendimento com opções de compressão de imagem RPCL (somente sem perdas) • 1.2.840.10008.1.2.4.203 — Compressão de imagem JPEG 2000 de alto rendimento	16 de fevereiro de 2024
Exemplos de código testados	HealthImaging a documenta ção fornece exemplos de códigos testados AWS SDKs para AWS CLI e para Python JavaScript, Java e C++. Para obter mais informações, consulte Exemplos de código.	19 de dezembro de 2023

Aumento do número máximo de arquivos para importações	HealthImaging suporta até 5.000 arquivos para um único trabalho de importação. Para obter mais informações, consulte Cotas de serviço.	19 de dezembro de 2023
Pastas aninhadas para importações	HealthImaging suporta até 10.000 pastas aninhadas para um único trabalho de importação. Para obter mais informações, consulte Cotas de serviço.	1.º de dezembro de 2023
Importações mais rápidas	HealthImaging fornece importações 20 vezes mais rápidas em todas as regiões suportadas. Para obter mais informações, consulte Endpoints de serviço.	1.º de dezembro de 2023
CloudFormation apoio	HealthImaging oferece suporte à infraestrutura como código (IaC) para provisionar armazenamentos de dados. Para obter mais informações, consulte Criação de HealthImaging recursos com CloudFormation.	21 de setembro de 2023

Disponibilidade geral	HealthImaging A AWS está disponível para todos os clientes nas regiões Leste dos EUA (Norte da Virgínia), Oeste dos EUA (Oregon), Europa (Irlanda) e Ásia-Pacífico (Sydney). Para obter mais informações, consulte Endpoints de serviço.	26 de julho de 2023

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.