aws

Guía para desarrolladores

AWS HealthImaging

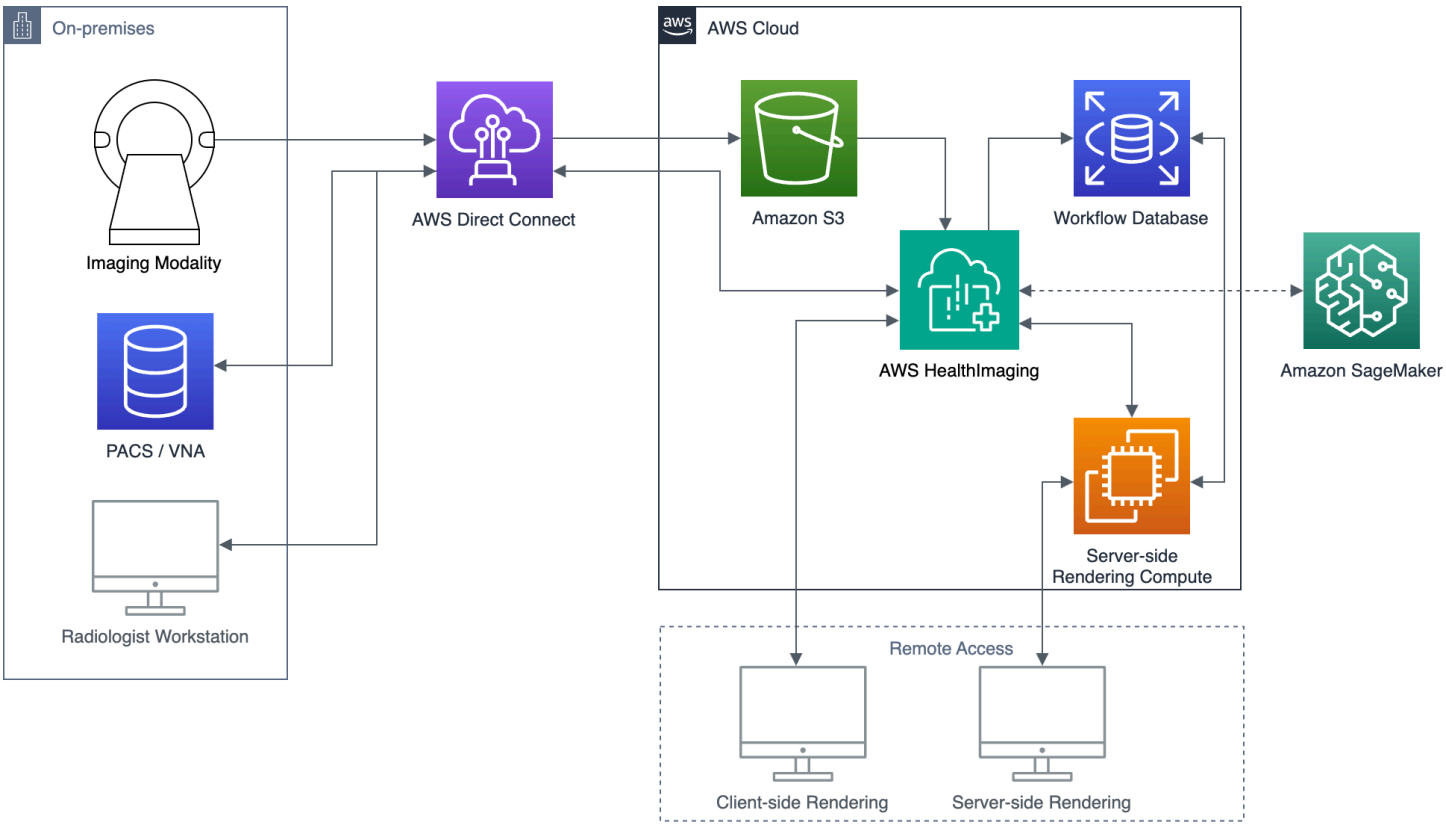
AWS HealthImaging: Guía para desarrolladores

Table of Contents

¿Qué es AWS HealthImaging?

AWS HealthImaging es un servicio compatible con la HIPAA que permite a los proveedores de atención médica, las organizaciones de ciencias de la vida y sus socios de software almacenar, analizar y compartir imágenes médicas en la nube a una escala de petabytes. HealthImaginglos casos de uso incluyen:

- Imágenes empresariales: almacene y transmita sus datos de imágenes médicas directamente desde la AWS nube y, al mismo tiempo, conserve el rendimiento de baja latencia y la alta disponibilidad.
- Archivo de imágenes a largo plazo: ahorre costes en el archivado de imágenes a largo plazo y, al mismo tiempo, mantenga el acceso a la recuperación de imágenes en menos de un segundo.
- Desarrollo de inteligencia artificial y aprendizaje automático (AI/ML): ejecute inferencias de inteligencia artificial y aprendizaje automático (AI/ML) sobre su archivo de imágenes con la ayuda de otras herramientas y servicios.
- Análisis multimodal: combine sus datos de imágenes clínicas con AWS HealthLake (datos de salud) y AWS HealthOmics (datos ómicos) para obtener información útil para la medicina de precisión.



AWS HealthImaging proporciona acceso a datos de imágenes (por ejemplo, rayos X, tomografía computarizada, resonancia magnética o ultrasonido) para que las aplicaciones de imágenes médicas integradas en la nube puedan alcanzar el rendimiento que antes solo era posible en las instalaciones. De este HealthImaging modo, usted reduce los costes de infraestructura al ejecutar sus aplicaciones de diagnóstico por imágenes médicas a escala a partir de una única copia fidedigna de cada imagen médica. Nube de AWS

Temas

Aviso importante

AWS no HealthImaging sustituye el asesoramiento, el diagnóstico o el tratamiento de un médico profesional y no pretende curar, tratar, mitigar, prevenir ni diagnosticar ninguna enfermedad o afección de salud. Usted es responsable de instituir la revisión humana como parte de cualquier uso de AWS HealthImaging, incluso en asociación con cualquier producto de terceros destinado a fundamentar la toma de decisiones clínicas. AWS solo HealthImaging debe utilizarse en la atención de pacientes o en situaciones clínicas tras su revisión por parte de profesionales médicos cualificados que apliquen un criterio médico sólido.

Características de AWS HealthImaging

AWS HealthImaging ofrece las siguientes funciones.

Metadatos DICOM fáciles de usar para desarrolladores

AWS HealthImaging simplifica el desarrollo de aplicaciones al devolver los metadatos DICOM en un formato fácil de usar para los desarrolladores. Tras importar los datos de las imágenes, podrá acceder a los atributos de los metadatos individuales utilizando palabras clave fáciles de usar en lugar de números hexadecimales de grupos o elementos desconocidos. Los elementos DICOM a nivel de paciente, estudio y serie están normalizados, de modo que no es necesario que los desarrolladores de aplicaciones se ocupen de las inconsistencias entre las instancias de SOP. Además, se puede acceder directamente a los valores de los atributos de los metadatos en los tipos de tiempo de ejecución nativos.

Decodificación de imágenes acelerada por SIMD

AWS HealthImaging devuelve marcos de imagen (datos de píxeles) codificados con JPEG 2000 (HTJ2K) de alto rendimiento, un códec de compresión de imágenes avanzado. HTJ2K aprovecha las ventajas de los datos múltiples con una sola instrucción (single instruction multiple data, SIMD) de los procesadores modernos para ofrecer nuevos niveles de rendimiento. HTJ2K es un orden de magnitud más rápido que el JPEG2000 y, como mínimo, el doble de rápido que todas las demás sintaxis de transferencia DICOM. WASM-SIMD se puede utilizar para llevar esta velocidad extrema a los visores web que no ocupan espacio.

Verificación de datos de píxeles

AWS HealthImaging proporciona una verificación de datos de píxeles integrada al comprobar el estado de codificación y decodificación sin pérdidas de cada imagen durante la importación. Para obtener más información, consulte Verificación de datos de píxeles.

Rendimiento líder del sector

AWS HealthImaging establece un nuevo estándar de rendimiento de carga de imágenes gracias a su eficiente codificación de metadatos, compresión sin pérdidas y acceso a datos de resolución progresiva. Gracias a la codificación eficiente de los metadatos, los visualizadores de imágenes y los algoritmos de IA comprenden el contenido de los estudios DICOM sin tener que cargar la información de la imagen. La compresión avanzada de imágenes hace que estas se carguen más rápido sin que ello comprometa la calidad de la imagen. Además, la resolución progresiva permite cargar las imágenes de las miniaturas, las regiones de interés y los dispositivos móviles de baja resolución aún más rápido.

Importaciones DICOM escalables

HealthImaging Las importaciones de AWS aprovechan las modernas tecnologías nativas de la nube para importar varios estudios DICOM en paralelo. Los archivos históricos se pueden importar rápidamente sin que ello afecte a las cargas de trabajo clínicas de los nuevos datos. Para más información sobre las instancias de SOP compatibles y las sintaxis de transferencia, consulte DICOMapoyo.

Servicios relacionados AWS

AWS HealthImaging presenta una estrecha integración con otros AWS servicios. Es útil conocer los siguientes servicios para aprovecharlos al máximo HealthImaging.

- AWS Identity and Access Management— Utilice la IAM para gestionar de forma segura las identidades y el acceso a HealthImaging los recursos.
- Amazon Simple Storage Service: utilice Amazon S3 como área de almacenamiento provisional para importar datos DICOM. HealthImaging
- Amazon CloudWatch: se usa CloudWatch para observar y monitorear HealthImaging los recursos.
- AWS CloudTrail— Se utiliza CloudTrail para realizar un seguimiento de la actividad HealthImaging de los usuarios y el uso de la API.
- AWS CloudFormation— Úselo AWS CloudFormation para implementar plantillas de infraestructura como código (IaC) para crear recursos. HealthImaging

- AWS PrivateLink— Utilice Amazon VPC para establecer la conectividad entre Amazon Virtual Private Cloud HealthImaging y Amazon sin exponer los datos a Internet.
- Amazon EventBridge: utilícelo EventBridge para crear aplicaciones escalables y basadas en eventos mediante la creación de reglas que dirijan HealthImaging los eventos a los objetivos.

Acceso a AWS HealthImaging

Puede acceder a AWS HealthImaging mediante los AWS Management Console, AWS Command Line Interface y los AWS SDK. Esta guía proporciona instrucciones de procedimiento AWS Management Console y ejemplos de código para los AWS SDK AWS CLI y los SDK.

AWS Management Console

AWS Management Console Proporciona una interfaz de usuario basada en la web para administrar los recursos HealthImaging y sus recursos asociados. Si ha creado una AWS cuenta, puede iniciar sesión en la HealthImaging consola.

AWS Command Line Interface (AWS CLI)

AWS CLI Proporciona comandos para un amplio conjunto de AWS productos y es compatible con Windows, Mac y Linux. Para obtener más información, consulte la Guía del usuario de AWS Command Line Interface .

AWS SDK

AWS Los SDK proporcionan bibliotecas, ejemplos de código y otros recursos para los desarrolladores de software. Estas bibliotecas proporcionan funciones básicas que automatizan tareas como la firma criptográfica de las solicitudes, el reintento de las solicitudes o el tratamiento de las respuestas de error. Para obtener más información, consulte Herramientas para crear en AWS.

Solicitudes HTTP

Puede HealthImaging realizar acciones mediante solicitudes HTTP, pero debe especificar distintos puntos de conexión en función del tipo de acciones que se utilicen. Para obtener más información, consulte APIAcciones compatibles para las solicitudes HTTP.

Cumplimiento con para la HIPAA y seguridad de la información

Se trata de un servicio compatible con HIPAA. Para obtener más información sobre AWS la Ley de Portabilidad y Responsabilidad de los Seguros de Salud de los Estados Unidos de 1996 (HIPAA) y el uso de AWS los servicios para procesar, almacenar y transmitir información de salud protegida (PHI), consulte Descripción general de la HIPAA.

Las conexiones que HealthImaging contienen la PHI y la información de identificación personal (PII) deben estar cifradas. De forma predeterminada, todas las conexiones deben HealthImaging usar HTTPS a través de TLS. HealthImaging almacena el contenido cifrado de los clientes y funciona de acuerdo con el modelo de responsabilidad AWS compartida.

Para más información sobre la conformidad, consulte Validación de la conformidad de AWS HealthImaging.

Precios

HealthImaging le ayuda a automatizar la gestión del ciclo de vida de los datos clínicos con una organización inteligente en niveles. Para obtener más información, consulte Descripción de los niveles de almacenamiento.

Para obtener información general sobre los precios, consulte los HealthImaging precios de AWS. Para calcular los costos, utilice la calculadora de HealthImaging precios de AWS.

Cómo empezar a usar AWS HealthImaging

Para empezar a usar AWS HealthImaging, configure una AWS cuenta y cree un AWS Identity and Access Management usuario. Para utilizar la AWS CLI o los SDK de AWS, debe instalarlos y configurarlos.

Tras aprender HealthImaging los conceptos y la configuración, dispondrá de un breve tutorial con ejemplos de código que le ayudará a empezar.

Temas

- AWS HealthImaging: conceptos básicos y terminología
- Configuración AWS HealthImaging
- AWS HealthImaging tutorial

AWS HealthImaging: conceptos básicos y terminología

Los términos y conceptos que se detallan a continuación son fundamentales para entender y utilizar AWS HealthImaging.

Conceptos

- Almacén de datos
- Conjuntos de imágenes
- Metadatos
- Marcos de imágenes

Almacén de datos

Los almacenes de datos son repositorios de datos de imágenes médicas que se encuentran en una sola Región de AWS. Las cuentas de AWS pueden varios almacenes de datos o ninguno. Cada almacén de datos tiene su propia clave de cifrado AWS KMS, por lo que los datos de un almacén de datos se pueden aislar física y lógicamente de los datos de otros almacenes de datos. Los almacenes de datos permiten controlar el acceso mediante roles de IAM, permisos y un control de acceso basado en atributos.

Para obtener más información, consulte Administración de almacenes de datos y Descripción de los niveles de almacenamiento.

Conjuntos de imágenes

Los conjuntos de imágenes son un concepto de AWS que define un mecanismo de agrupación abstracto para optimizar los datos de imágenes médicas relacionadas. Al importar los datos de DICOM P10 a AWS HealthImaging, estos se transforman en conjuntos de imágenes compuestos por metadatos y marcos de imágenes (datos de píxeles). La importación de datos DICOM P10 da como resultado conjuntos de imágenes que contienen metadatos DICOM y marcos de imágenes para una o más instancias de pares de objetos y servicios (SOP) de la misma serie DICOM.

Para obtener más información, consulte Importación de datos de imágenes y Descripción de los conjuntos de imágenes.

Metadatos

Los metadatos son los atributos que no son píxeles de los conjuntos de imágenes. En el caso del DICOM, esto incluye los datos demográficos de los pacientes, los detalles del procedimiento y otros parámetros específicos de la adquisición. AWS HealthImaging separa el conjunto de imágenes en metadatos y marcos de imágenes (datos de píxeles) para que las aplicaciones puedan acceder al conjunto rápidamente. Esto resulta útil para los visores de imágenes, los análisis y los casos de uso de inteligencia artificial y aprendizaje automático que no necesitan la información de los píxeles. Los datos DICOM se normalizan a nivel de paciente, estudio y serie, lo que elimina las incoherencias. Esto simplifica el uso de los datos, aumenta la seguridad y mejora el rendimiento del acceso.

Para obtener más información, consulte Obtención de metadatos de conjuntos de imágenes y Normalización de metadatos.

Marcos de imágenes

Los conjuntos de imágenes son la información de los píxeles existentes en un conjunto de imágenes y que forman una imagen médica en 2D. Durante la importación, AWS HealthImaging codifica todos los marcos de imagen en JPEG 2000 (HTJ2K) de alto rendimiento. Por lo tanto, los marcos de las imágenes deben decodificarse antes de visualizarlos.

Para obtener más información, consulte Obtención de datos de píxeles de los conjuntos de imágenes y Bibliotecas de decodificación HTJ2K.

Configuración AWS HealthImaging

Debe configurar su AWS entorno antes de usarlo AWS HealthImaging. Antes de completar el tutorial de la sección siguiente, debe haber los temas que se detallan a continuación.

Temas

- Inscríbase en un Cuenta de AWS
- Creación de un usuario con acceso administrativo
- Crear buckets de S3
- Crear un almacén de datos
- Cree un IAM usuario con permiso de acceso HealthImaging total
- Cree un IAM rol para la importación
- Instale la AWS CLI (opcional)

Inscríbase en un Cuenta de AWS

Si no tienes un Cuenta de AWS, complete los pasos siguientes para crear uno.

Para suscribirte a una Cuenta de AWS

1. Abrir https://portal.aws.amazon.com/billing/registro.
2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

   Cuando te registras en una Cuenta de AWS, un Usuario raíz de la cuenta de AWSse crea. El usuario root tiene acceso a todos Servicios de AWS y los recursos de la cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar tareas que requieren acceso de usuario raíz.

AWS te envía un correo electrónico de confirmación una vez finalizado el proceso de registro. En cualquier momento, puede ver la actividad de su cuenta actual y administrarla accediendo a https://aws.amazon.com/y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de suscribirse a una Cuenta de AWS, asegure su Usuario raíz de la cuenta de AWS, habilitar AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Consolecomo propietario de la cuenta seleccionando el usuario root e introduciendo su Cuenta de AWS dirección de correo electrónico. En la siguiente página, escriba su contraseña.

   Para obtener ayuda para iniciar sesión con un usuario root, consulte Iniciar sesión como usuario root en AWS Sign-In Guía del usuario.
2. Activa la autenticación multifactorial (MFA) para tu usuario root.

   Para obtener instrucciones, consulte Habilitar un MFA dispositivo virtual para su Cuenta de AWS usuario root (consola) en la Guía IAM del usuario.

Creación de un usuario con acceso administrativo

1. Habilite IAM Identity Center.

   Para obtener instrucciones, consulte Habilitar AWS IAM Identity Center en la AWS IAM Identity Center Guía del usuario.
2. En IAM Identity Center, conceda acceso administrativo a un usuario.

   Para ver un tutorial sobre el uso de Directorio de IAM Identity Center como fuente de identidad, consulte Configurar el acceso de los usuarios con la configuración predeterminada Directorio de IAM Identity Center en la AWS IAM Identity Center Guía del usuario.

Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con su usuario de IAM Identity Center, utilice el inicio de sesión URL que se envió a su dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

  Para obtener ayuda para iniciar sesión con un usuario de IAM Identity Center, consulte Iniciar sesión en el AWS acceda al portal en el AWS Sign-In Guía del usuario.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos con privilegios mínimos.

   Para obtener instrucciones, consulte Crear un conjunto de permisos en el AWS IAM Identity Center Guía del usuario.

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

   Para obtener instrucciones, consulte Añadir grupos en AWS IAM Identity Center Guía del usuario.

Crear buckets de S3

Para importar datos de DICOM P10 AWS HealthImaging, se recomiendan dos depósitos de Amazon S3. El depósito de entrada de Amazon S3 almacena los datos del DICOM P10 que se van a importar y HealthImaging lee de este depósito. El bucket de salida de Amazon S3 almacena los resultados del procesamiento del trabajo de importación y HealthImaging escribe en este bucket. Encontrará una representación gráfica de todo ello en el diagrama Introducción a los trabajos de importación.

> **Note**
>
> Debido a AWS Identity and Access Management (IAM), los nombres de sus buckets de Amazon S3 deben ser únicos. Para más información, consulte Reglas de nomenclatura de buckets en la Guía del usuario de Amazon Simple Storage Service.

A los efectos de esta guía, especificamos los siguientes buckets de entrada y salida de Amazon S3 en la IAMfunción de importación.

- Bucket de entrada: `arn:aws:s3:::medical-imaging-dicom-input`
- Bucket de salida: `arn:aws:s3:::medical-imaging-output`

Para más información, consulte Crear un bucket en la Guía del usuario de Amazon S3.

Crear un almacén de datos

Al importar los datos de imágenes médicas, el almacén de AWS HealthImaging datos contiene los resultados de los archivos DICOM P10 transformados, que se denominan conjuntos de imágenes. Encontrará una representación gráfica de todo ello en el diagrama Introducción a los trabajos de importación.

> **Tip**
>
> Se genera un `datastoreID` al crear un almacén de datos. Debe utilizar el `datastoreID` al completar la trust relationship para importar más adelante en esta sección.

Para crear un almacén de datos, consulte Creación de un almacén de datos.

Cree un IAM usuario con permiso de acceso HealthImaging total

> **Práctica recomendada**
>
> Le sugerimos que cree IAM usuarios independientes para diferentes necesidades, como la importación, el acceso a los datos y la administración de datos. Esto se corresponde con conceder el acceso con privilegios mínimos en el AWS Well-Architected Framework.
> Para los fines del tutorial de la siguiente sección, utilizará un solo IAM usuario.

Para crear un usuario de IAM

1. Siga las instrucciones para crear un IAM usuario en su AWS cuenta en la Guía IAM del usuario. Considere asignar un nombre al usuario `ahiadmin` (o similar) a efectos de clarificación.
2. Asigne la política `AWSHealthImagingFullAccess` gestionada al IAM usuario. Para obtener más información, consulte Política administrada por AWS: AWSHealthImagingFullAccess.

   > **Note**
   >
   > IAMlos permisos se pueden restringir. Para obtener más información, consulte Políticas administradas por AWS para AWS HealthImaging.

Cree un IAM rol para la importación

> **Note**
>
> Las instrucciones siguientes se refieren a un AWS Identity and Access Management (IAM) rol que otorga acceso de lectura y escritura a los buckets de Amazon S3 para importar sus DICOM datos. Si bien el rol es obligatorio para el tutorial de la siguiente sección, le recomendamos que añada IAM permisos a los usuarios, grupos y roles que los utilicenPolíticas administradas por AWS para AWS HealthImaging, ya que son más fáciles de usar que escribir las políticas usted mismo.

Un IAM rol es una IAM identidad que puedes crear en tu cuenta y que tiene permisos específicos. Para iniciar un trabajo de importación, el IAM rol que realiza la `StartDICOMImportJob` acción debe estar asociado a una política de usuario que conceda acceso a los buckets de Amazon S3 que se utilizan para leer sus datos de DICOM P10 y almacenar los resultados del procesamiento del trabajo de importación. También se le debe asignar una relación de confianza (política) que le permita AWS HealthImaging asumir el rol.

Para crear un IAM rol con fines de importación

1. Con la IAMconsola, cree un rol denominado`ImportJobDataAccessRole`. Este rol se utiliza para el tutorial de la siguiente sección. Para obtener más información, consulte Creación de IAM roles en la Guía del IAM usuario.

   > **Tip**
   >
   > Para los fines de esta guía, los ejemplos de código Inicio de un trabajo de importación hacen referencia al `ImportJobDataAccessRole` IAM rol.

2. Adjunte una política de IAM permisos al IAM rol. Esta política de permisos otorga acceso a los buckets de entrada y salida de Amazon S3. Adjunte la siguiente política de permisos al IAM rol`ImportJobDataAccessRole`.

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

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

3. Adjunte la siguiente relación de confianza (política) al `ImportJobDataAccessRole` IAM rol. La política de confianza requiere el `datastoreId` que se generó al completar la sección Crear un almacén de datos. En el tutorial que sigue a este tema se asume que está utilizando un almacén de AWS HealthImaging datos, pero con buckets, IAM funciones y políticas de confianza de Amazon S3 específicos del almacén de datos.

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

```
            "Action": "sts:AssumeRole",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:SourceAccount": "accountId"
                },
                "ForAllValues:ArnEquals": {
                    "aws:SourceArn": "arn:aws:medical-
imaging:region:accountId:datastore/datastoreId"
                }
            }
        }
    ]
}
```

Para obtener más información sobre la creación y el uso de IAM políticas con AWS HealthImaging, consulte. Identity and Access Management para AWS HealthImaging

Para obtener más información sobre las IAM funciones en general, consulte las IAMfunciones en la Guía del IAM usuario. Para obtener más información sobre IAM las políticas y los permisos en general, consulte IAMPolíticas y permisos en la Guía del IAM usuario.

Instale la AWS CLI (opcional)

Se requiere el siguiente procedimiento si utiliza el AWS Command Line Interface. Si utilizas el AWS Management Console o AWS SDKs, puede omitir el siguiente procedimiento.

Para configurar el AWS CLI

1. Descargue y configure el AWS CLI. Para obtener instrucciones, consulte los siguientes temas en la AWS Command Line Interface Guía del usuario.

   - Instalación o actualización de la última versión de AWS CLI
   - Cómo empezar con el AWS CLI

2. En el navegador AWS CLI `config`archivo, añada un perfil con nombre para el administrador. Este perfil se utiliza cuando se ejecuta el AWS CLI comandos. Según el principio de seguridad del privilegio mínimo, le recomendamos que cree un IAM rol independiente con privilegios específicos para las tareas que se estén realizando. Para obtener más información sobre los perfiles con nombre asignado, consulte los ajustes de configuración y de los archivos de credenciales en la AWS Command Line Interface Guía del usuario.

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

3. Verifique la configuración con el comando `help` siguiente.

```
aws medical-imaging help
```

Si el archivo de AWS CLI está configurado correctamente, verá una breve descripción AWS HealthImaging y una lista de los comandos disponibles.

AWS HealthImaging tutorial

Objetivo

El objetivo de este tutorial es importar binarios (`.dcm`archivos) de DICOM P10 a un almacén de AWS HealthImaging datos y transformarlos en conjuntos de imágenes compuestos por metadatos y marcos de imágenes (datos de píxeles). Tras importar los DICOM datos, utilice acciones nativas de HealthImaging la nube para acceder a los conjuntos de imágenes, los metadatos y los marcos de imágenes según sus preferencias de acceso.

Requisitos previos

Todos los procedimientos que se enumeran en Configuración son necesarios para completar este tutorial.

Pasos del tutorial

1. Iniciar el trabajo de importación
2. Obtener las propiedades del trabajo de importación
3. Buscar conjuntos de imágenes
4. Obtener las propiedades de los conjuntos de imágenes
5. Obtener metadatos del conjunto de imágenes
6. Obtener datos de píxeles del conjunto de imágenes
7. Eliminación de almacenes de datos

Administrar almacenes de datos con AWS HealthImaging

Con AWS HealthImaging él, puede crear y administrar almacenes de datos para recursos de imágenes médicas. En los siguientes temas se describe cómo utilizar las acciones nativas de la HealthImaging nube para crear, describir, enumerar y eliminar almacenes de datos mediante las AWS Management Console teclas AWS CLI, y AWS SDKs.

> **Note**
>
> El último tema de este capítulo trata sobre cómo entender los niveles de almacenamiento. Después de importar los datos de imágenes médicas a un almacén de HealthImaging datos, estos se mueven automáticamente entre dos niveles de almacenamiento en función del tiempo y el uso. Los niveles de almacenamiento tienen distintos niveles de precios, por lo que es importante entender el proceso de traslado de los niveles y los HealthImaging recursos que se reconocen a efectos de facturación.

Temas

Creación de un almacén de datos

Utilice la `CreateDatastore` acción para crear un banco AWS HealthImaging de datos para importar archivos DICOM P10. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `CreateDatastore`la AWS HealthImaging APIReferencia.

> **Importante**
>
> No nombre los almacenes de datos que contengan información de salud protegida (PHI), información de identificación personal (PII) u otra información confidencial o delicada.

Cómo crear un almacén de datos

Elija un menú según sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de creación de un almacén de datos de la HealthImaging consola.
2. Introduzca un nombre para el almacén de datos en Nombre del almacén de datos, en la sección Detalles.
3. En Cifrado de datos, elija una AWS KMS clave para cifrar los recursos. Para obtener más información, consulte Protección de datos en AWS HealthImaging.
4. En Etiquetas (opcional), podrá agregar etiquetas al almacén de datos cuando lo cree. Para obtener más información, consulte Etiquetado de un recurso.
5. Elija Crear almacén de datos.

AWS CLI y SDKs

Bash

AWS CLI con 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 API obtener más información, consulte CreateDatastoreen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Creación de un almacén de datos

En el siguiente ejemplo de código `create-datastore` se crea un almacén de datos con el nombre `my-datastore`.

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

Salida:

```
{
    "datastoreId": "12345678901234567890123456789012",
```

```
    "datastoreStatus": "CREATING"
}
```

Para obtener más información, consulte Crear un banco de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte CreateDatastoreen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte CreateDatastoreen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte CreateDatastoreen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en CreateDatastoreAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Obtención de propiedades de los almacenes de datos

Utilice la `GetDatastore` acción para recuperar las propiedades del almacén de AWS HealthImaging datos. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `GetDatastore`la AWS HealthImaging APIReferencia.

Cómo obtener propiedades de los almacenes de datos

Elija un menú según sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de Detalles del almacén de datos. Todas las propiedades del almacén de datos están disponibles en la sección Detalles. Para ver los conjuntos de datos asociados, las importaciones y las etiquetas, seleccione la pestaña correspondiente.

AWS CLI y SDKs

Bash

AWS CLI con 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 API obtener más información, consulte GetDatastoreen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Obtención de las propiedades de un almacén de datos

En el siguiente ejemplo de código `get-datastore` se obtienen las propiedades de un almacén de datos.

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

Salida:

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

Para obtener más información, consulte Obtener las propiedades del almacén de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetDatastoreen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetDatastoreen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte GetDatastoreen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en GetDatastoreAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Enumeración de almacenes de datos

Utilice la `ListDatastores` acción para enumerar los almacenes de datos disponibles en AWS HealthImaging. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `ListDatastores`la AWS HealthImaging APIReferencia.

Cómo enumerar almacenes de datos

Elija un menú según sus preferencias de acceso a AWS HealthImaging.

AWS Consola

- Abra la página de almacenes de datos de la HealthImaging consola.

  Todos los almacenes de datos aparecen en la sección almacenes de datos.

AWS CLI y SDKs

Bash

AWS CLI con 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 API obtener más información, consulte ListDatastoresen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Enumeración de almacenes de datos

En el siguiente ejemplo de código `list-datastores` se enumeran los almacenes de datos disponibles.

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

Salida:

```
{
    "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 obtener más información, consulte Listar los almacenes de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte ListDatastoresen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte ListDatastoresen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte ListDatastoresen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en ListDatastoresAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Eliminación de almacenes de datos

Utilice la `DeleteDatastore` acción para eliminar un banco AWS HealthImaging de datos. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `DeleteDatastore`la AWS HealthImaging APIReferencia.

> **Note**
>
> Para poder eliminar un almacén de datos, debe haber eliminado antes todos los conjuntos de imágenes que contiene. Para obtener más información, consulte Eliminación de un conjunto de imágenes.

Cómo eliminar un almacén de datos

Elija un menú según sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.
3. Elija Eliminar.

   Se abrirá la página de Eliminar almacén de datos.

4. Para confirmar la eliminación del almacén de datos, escriba el nombre del almacén de datos en el campo de entrada de texto.
5. Elija Eliminar almacén de datos.

AWS CLI y SDKs

Bash

AWS CLI con 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 API obtener más información, consulte DeleteDatastoreen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Eliminación de un almacén de datos

En el siguiente ejemplo de código `delete-datastore` se elimina un almacén de datos.

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

Salida:

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

Para obtener más información, consulte Eliminar un banco de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte DeleteDatastoreen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte DeleteDatastoreen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte DeleteDatastoreen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en DeleteDatastoreAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Descripción de los niveles de almacenamiento

AWS HealthImaging utiliza la organización inteligente en niveles para la administración automática del ciclo de vida clínico. Esto se traduce en un rendimiento y un precio atractivos tanto para los datos nuevos o activos como para los datos archivados a largo plazo sin ningún problema. HealthImaging factura el almacenamiento por GB al mes mediante los siguientes niveles.

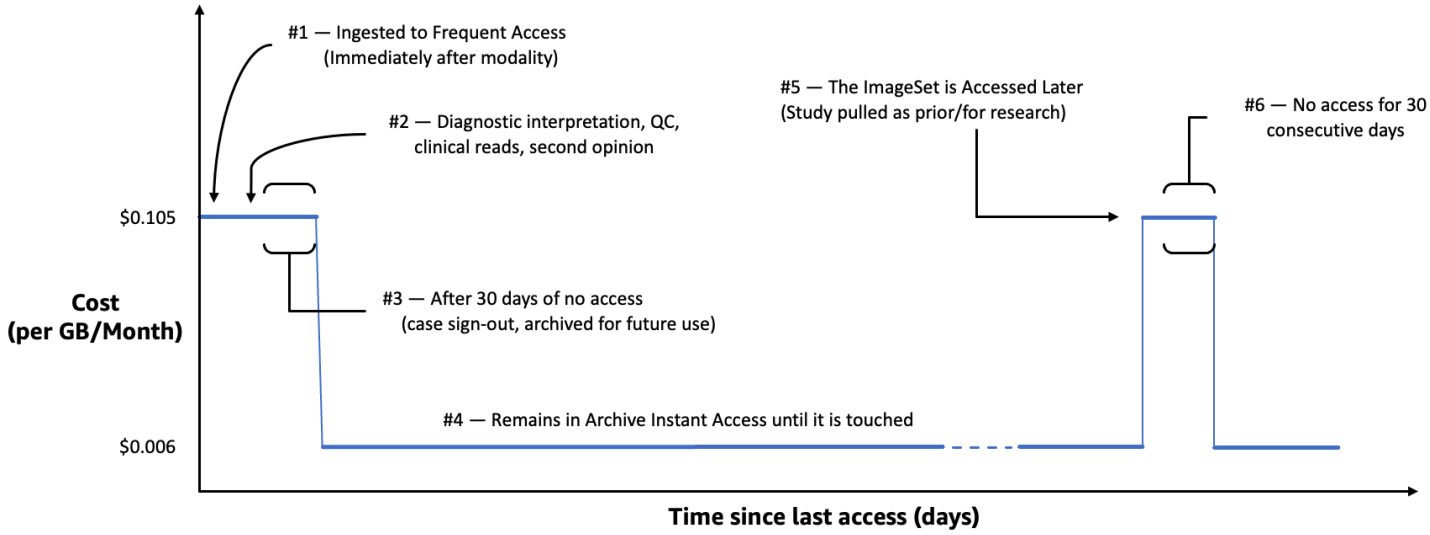
- Nivel de Acceso frecuente: un nivel para los datos a los que se accede con frecuencia.
- Nivel de Acceso instantáneo a los archivos: un nivel para los datos archivados.

> **Note**
>
> No hay ninguna diferencia de rendimiento entre los niveles Frequent Access y Archive Instant Access. La clasificación inteligente por niveles se aplica a acciones específicas de los conjuntos de imágenes de la API. La clasificación inteligente por niveles no reconoce las acciones de la API de almacenamiento, importación y etiquetado de datos. El paso de un nivel a otro es automático, depende del uso de la API, y se explica en la siguiente sección.

¿Cómo funciona el paso de un nivel a otro?

- Después de la importación, los conjuntos de imágenes comienzan en el nivel de Acceso frecuente.
- Tras 30 días consecutivos sin interacciones, pasan automáticamente al nivel de Acceso instantáneo de archivos.
- Los conjuntos de imágenes solo pueden ir del nivel de Acceso instantáneo de archivos al de Acceso frecuente si se interactúa con ellos.

El siguiente gráfico proporciona una visión general del proceso de organización HealthImaging inteligente por niveles.



¿Qué se considera una interacción?

Un toque es un acceso a una API específica a través de los AWS Management Console AWS CLI, o AWS los SDK y se produce cuando:

1. Se crea (`StartDICOMImportJob` o `CopyImageSet`) un nuevo conjunto de imágenes
2. Se actualiza (`UpdateImageSetMetadata` o `CopyImageSet`) un conjunto de imágenes
3. Se leen (`GetImageSetMetaData` o `GetImageFrame`) los metadatos o los marcos de imagen (datos de píxeles) asociados a un conjunto de imágenes.

Las siguientes acciones de la HealthImaging API permiten tocar y mover conjuntos de imágenes del nivel de acceso instantáneo de Archive al nivel de acceso frecuente.

- `StartDICOMImportJob`
- `GetImageSetMetadata`

- `GetImageFrame`
- `CopyImageSet`
- `UpdateImageSetMetadata`

> **Note**
>
> Aunque los marcos de imagen (datos de píxeles) no se pueden eliminar mediante la acción `UpdateImageSetMetadata`, se tienen en cuenta a efectos de facturación.

Las siguientes acciones de la HealthImaging API no dan lugar a modificaciones. Por lo tanto, no mueven los conjuntos de imágenes del nivel de Acceso instantáneo de archivos al nivel de Acceso frecuente.

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

Importación de datos de imágenes con AWS HealthImaging

La importación es el proceso de mover los datos de imágenes médicas de un depósito de entrada de Amazon S3 a un almacén de AWS HealthImaging datos. Durante la importación, AWS HealthImaging comprueba los datos en píxeles antes de transformar los archivos DICOM P10 en conjuntos de imágenes compuestos por metadatos y marcos de imágenes (datos en píxeles).

> **Importante**
>
> HealthImaging Los trabajos de importación procesan binarios (`.dcm`archivos) de DICOM instancias y los transforman en conjuntos de imágenes. Utilice acciones nativas de HealthImaging la nube (APIs) para administrar los almacenes de datos y los conjuntos de imágenes. Utilice HealthImaging la representación de los DICOMweb servicios para devolver DICOMweb las respuestas.

En los siguientes temas se describe cómo importar los datos de imágenes médicas a un banco de HealthImaging datos mediante AWS Management Console AWS CLI, y AWS SDKs.

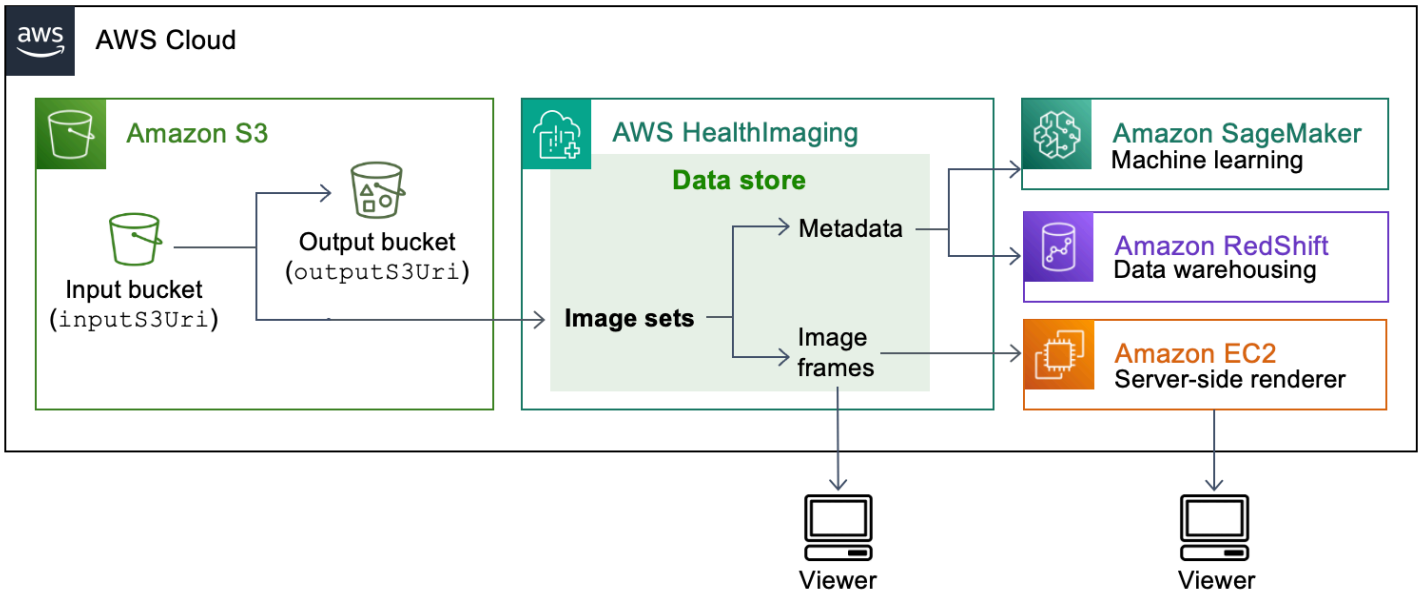
Temas

Introducción a los trabajos de importación

Tras crear un almacén de datos en AWS HealthImaging, debe importar los datos de imágenes médicas de su depósito de entrada de Amazon S3 a su almacén de datos para crear conjuntos de imágenes. Puede usar los AWS Management Console AWS SDK y los SDK para iniciar, describir y enumerar los trabajos de importación. AWS CLI

El siguiente diagrama proporciona una descripción general de cómo se HealthImaging importan los datos DICOM a un banco de datos y los transforma en conjuntos de imágenes. Los resultados del procesamiento de los trabajos de importación se almacenan en el bucket de salida de Amazon S3

(`outputS3Uri`) y los conjuntos de imágenes se almacenan en el almacén de HealthImaging datos de AWS.



Tenga en cuenta los siguientes puntos al importar sus archivos de imágenes médicas de Amazon S3 a un almacén de HealthImaging datos de AWS:

- Se admiten clases de SOP específicas y sintaxis de transferencia para los trabajos de importación. Para obtener más información, consulte DICOMapoyo.
- Durante la importación, se aplican restricciones de longitud a elementos DICOM específicos. Para que la importación sea correcta, asegúrese de que sus datos de imágenes médicas no superen las restricciones de longitud máxima. Para obtener más información, consulte DICOMrestricciones de elementos.
- Al principio de los trabajos de importación, se comprueba la verificación de los datos en píxeles. Para obtener más información, consulte Verificación de datos de píxeles.
- Hay puntos finales, cuotas y límites de regulación asociados a las acciones de importación. HealthImaging Para obtener más información, consulte Cuotas y puntos de conexión y Límites de limitación.
- Para cada trabajo de importación, los resultados del procesamiento se almacenan en la ubicación `outputS3Uri`. Los resultados del procesamiento se organizan en un archivo `job-output-manifest.json` y en las carpetas `SUCCESS` y `FAILURE`.

> **Note**
>
> Puede incluir hasta 10 000 carpetas anidadas para un solo trabajo de importación.

- El archivo `job-output-manifest.json` contiene el resultado de salida `jobSummary` e información adicional sobre los datos procesados. El ejemplo siguiente muestra la salida de un archivo `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/",
        "numberOfScannedFiles": 5,
        "numberOfImportedFiles": 3,
        "numberOfFilesWithCustomerError": 2,
        "numberOfFilesWithServerError": 0,
        "numberOfGeneratedImageSets": 2,
        "imageSetsSummary": [{
                "imageSetId": "12345612345612345678907890789012",
                "numberOfMatchedSOPInstances": 2
            },
            {
                "imageSetId": "123456123456123456789017891789012",
                "numberOfMatchedSOPInstances": 1
            }
        ]
    }
}
```

- La carpeta SUCCESS contiene el archivo `success.ndjson` con los resultados de todos los archivos de imágenes que se importaron correctamente. El ejemplo siguiente muestra la salida de un archivo `success.ndjson`.

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

- La carpeta FAILURE contiene el archivo `failure.ndjson` con los resultados de todos los archivos de imágenes que no se importaron correctamente. El ejemplo siguiente muestra la salida de un archivo `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"}}
```

- Los trabajos de importación se conservan en la lista de trabajos durante 90 días y, a continuación, se archivan.

Inicio de un trabajo de importación

Usa la `StartDICOMImportJob` acción para iniciar una verificación de datos de píxeles y una importación masiva de datos a un almacén de AWS HealthImaging datos. El trabajo de importación importa archivos DICOM P10 ubicados en el depósito de entrada de Amazon S3 especificado por el `inputS3Uri` parámetro. Los resultados del procesamiento del trabajo de importación se almacenan en el bucket de salida de Amazon S3 especificado por el parámetro `outputS3Uri`.

> **Note**
>
> HealthImaging admite la importación de datos desde buckets de Amazon S3 ubicados en otras regiones compatibles. Para lograr esta funcionalidad, proporcione el `inputOwnerAccountId` parámetro al iniciar un trabajo de importación. Para obtener más información, consulte Importación multicuenta para AWS HealthImaging.

Durante la importación, las restricciones de longitud se aplican a DICOM elementos específicos. Para obtener más información, consulte DICOMrestricciones de elementos.

Los menús siguientes proporcionan un procedimiento para la AWS Management Console y ejemplos de código para el AWS CLI y AWS SDKs. Para obtener más información, consulte `StartDICOMImportJob`la AWS HealthImaging APIReferencia.

Cómo iniciar un trabajo de importación

Elija un menú según sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.
3. Selecciona Importar DICOM datos.

   Se abre la página Importar DICOM datos.
4. En la sección Detalles, introduce la siguiente información:
   - Nombre (opcional)
   - Importar la ubicación de origen en S3
   - ID de cuenta del propietario del bucket de origen (opcional)
   - Clave de cifrado (opcional)
   - Destino de salida en S3
5. En la sección Acceso al servicio, elija Usar un rol de servicio existente y seleccione el rol en el menú del Nombre del rol de servicio o elija Crear y usar un nuevo rol de servicio.
6. Seleccione Importar.

AWS CLI y SDKs

C++

SDKpara 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 API obtener más información, consulte S tartDICOMImport Job en AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Inicio de un trabajo de importación DICOM

En el siguiente ejemplo de código `start-dicom-import-job` se inicia un trabajo de importación 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"
```

Salida:

```
{
```

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

Para obtener más información, consulte Iniciar un trabajo de importación en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte S tartDICOMImport Job en AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte S tartDICOMImport Job en AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte S tartDICOMImport Job en AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte S tartDICOMImport Job en AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Obtención de las propiedades de trabajos de importación

Utilice esta `GetDICOMImportJob` acción para obtener más información sobre la AWS HealthImaging importación de propiedades de un trabajo. Por ejemplo, después de iniciar un trabajo de importación, puede ejecutar `GetDICOMImportJob` para buscar el estado del trabajo. Cuando el `jobStatus` vuelve como `COMPLETED`, ya podrá acceder a sus conjuntos de imágenes.

> **Note**
>
> El `jobStatus` se refiere a la ejecución del trabajo de importación. Por lo tanto, un trabajo de importación puede devolver un `jobStatus` como `COMPLETED` incluso si se detectan problemas de validación durante la importación. Aunque un `jobStatus` vuelve como `COMPLETED`, le recomendamos que revise los manifiestos de salida escritos en Amazon S3, ya que proporcionan información sobre el éxito o el fracaso de las importaciones de objetos P10.

Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `GetDICOMImportJob`la AWS HealthImaging APIReferencia.

Cómo obtener las propiedades de los trabajos de importación

Elija un menú según sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de Detalles del almacén de datos. La pestaña Conjuntos de imágenes está seleccionada por defecto.

3. Seleccione la pestaña Importaciones.
4. Elija un trabajo de importación.

   Se abrirá la página de Detalles del trabajo de importación, que muestra las propiedades de los trabajos de importación.

AWS CLI y SDKs

C++

SDKpara 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 API obtener más información, consulte G etDICOMImport Job en AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Obtención de las propiedades de un trabajo de importación DICOM

En el siguiente ejemplo de código `get-dicom-import-job` se obtienen las propiedades de un trabajo de importación DICOM.

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

Salida:

```
{
    "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 obtener más información, consulte Obtener las propiedades de los trabajos de importación en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte G etDICOMImport Job en AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte G etDICOMImport Job en AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte G etDICOMImport Job en AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte G etDICOMImport Job en AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Enumeración de trabajos de importación

Utilice la `ListDICOMImportJobs` acción para enumerar los trabajos de importación creados para un banco HealthImaging de datos específico. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `ListDICOMImportJobs`la AWS HealthImaging APIReferencia.

> **Note**
>
> Los trabajos de importación se conservan en la lista de trabajos durante 90 días y, a continuación, se archivan.

Cómo enumerar trabajos de importación

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

Se abrirá la página de Detalles del almacén de datos. La pestaña Conjuntos de imágenes está seleccionada por defecto.

3. Seleccione la pestaña Importaciones para ver todos los trabajos de importación asociados.

AWS CLI y SDKs

CLI

AWS CLI

Enumeración de los trabajos de importación DICOM

En el siguiente ejemplo de código `list-dicom-import-jobs` se enumeran los trabajos de importación DICOM.

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

Salida:

```
{
    "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 obtener más información, consulte Listar los trabajos de importación en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte L istDICOMImport Jobs en AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte L istDICOMImport Jobs en AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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);

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

  return jobSummaries;
};
```

- Para API obtener más información, consulte L istDICOMImport Jobs en AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte L Jobs en istDICOMImport AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Acceder a conjuntos de imágenes con AWS HealthImaging

El acceso a los datos de imágenes médicas AWS HealthImaging normalmente implica buscar un conjunto de imágenes con una clave única y obtener los metadatos y los marcos de imágenes asociados (datos en píxeles).

> **Importante**
>
> Durante la importación, HealthImaging procesa los `.dcm` archivos binarios de las DICOM instancias y los transforma en conjuntos de imágenes. Utilice las acciones nativas de HealthImaging la nube (APIs) para gestionar los almacenes de datos y los conjuntos de imágenes. Utilice HealthImaging la representación de los DICOMweb servicios para devolver DICOMweb las respuestas.

En los siguientes temas se explica cómo utilizar las acciones nativas de la HealthImaging nube en AWS Management Console AWS CLI, y AWS SDKs cómo buscar conjuntos de imágenes y obtener sus propiedades, metadatos y marcos de imágenes asociados.

Temas

Descripción de los conjuntos de imágenes

Los conjuntos de imágenes son un AWS concepto que sirve de base para AWS HealthImaging. Los conjuntos de imágenes se crean al importar los datos de DICOM HealthImaging, por lo que es necesario conocerlos bien cuando se trabaja con el servicio.

Los conjuntos de imágenes se introdujeron por varias razones:

- Son compatibles con una amplia variedad de flujos de trabajo de imágenes médicas (clínicas y no clínicas) mediante API flexibles.

- Permiten maximizar la seguridad de los pacientes agrupando únicamente los datos relacionados.
- Fomentar la limpieza de los datos para ofrecer una mayor visibilidad de las incoherencias. Para obtener más información, consulte Modificación de conjuntos de imágenes.

> **Importante**
>
> El uso clínico de los datos DICOM antes de su limpieza puede ser perjudicial para el paciente.

Los siguientes menús describen los conjuntos de imágenes con más detalle y proporcionan ejemplos y diagramas para ayudarle a comprender su funcionalidad y propósito. HealthImaging

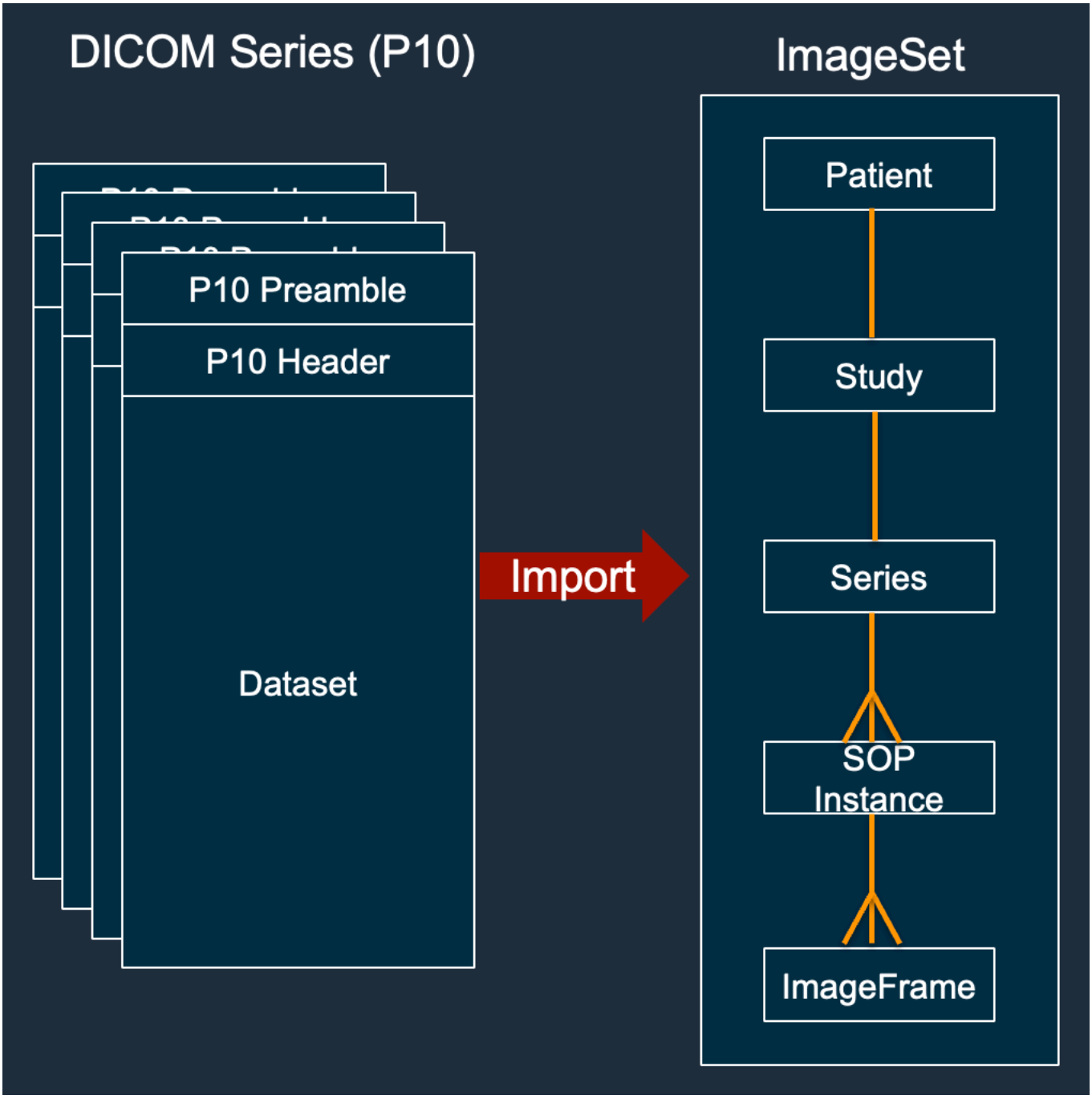
¿Qué son los conjuntos de imágenes?

Un conjunto de imágenes es un AWS concepto que define un mecanismo de agrupación abstracto para optimizar los datos de imágenes médicas relacionados. Al importar los datos de imágenes del DICOM P10 a un almacén de HealthImaging datos de AWS, se transforman en conjuntos de imágenes compuestos por metadatos y marcos de imágenes (datos de píxeles).

> **Note**
>
> Los metadatos de los conjuntos de imágenes están normalizados. Dicho de otra manera, un conjunto común de atributos y valores se asigna a los elementos de nivel de paciente, estudio y serie que figuran en Registry of DICOM Data Elements.
> Los marcos de imágenes (datos de píxeles) se codifican en formato JPEG 2000 de alto rendimiento (HTJ2K) y deben decodificarse para poder visualizarlos.

Los conjuntos de imágenes son AWS recursos, por lo que se les asignan nombres de recursos de Amazon (ARN). Se pueden etiquetar con hasta 50 pares de clave-valor y se les puede conceder control de acceso basado en roles (RBAC) y control de acceso basado en atributos (ABAC) mediante IAM. Además, los conjuntos de imágenes tienen control de versiones para conservar todos los cambios y poder acceder a las versiones anteriores.

La importación de datos DICOM P10 da como resultado conjuntos de imágenes que contienen metadatos DICOM y marcos de imágenes para una o más instancias de pares de objetos y servicios (SOP) de la misma serie DICOM.



Note

Trabajos de importación DICOM:

- Cree siempre nuevos conjuntos de imágenes, y nunca actualice los conjuntos de imágenes existentes.

- No desduplique el almacenamiento de las instancias SOP, ya que cada importación de la misma instancia SOP utiliza almacenamiento adicional.
- Puede crear varios conjuntos de imágenes para una sola serie DICOM. Por ejemplo, cuando hay una variante de un atributo de metadatos normalizado, como una `PatientName` discordancia.

¿Qué aspecto tienen los metadatos del conjunto de imágenes?

Utilice la `GetImageSetMetadata` acción para recuperar los metadatos del conjunto de imágenes. Los metadatos devueltos se comprimen con`gzip`, por lo que debe descomprimirlos antes de verlos. Para obtener más información, consulte Obtención de metadatos de conjuntos de imágenes.

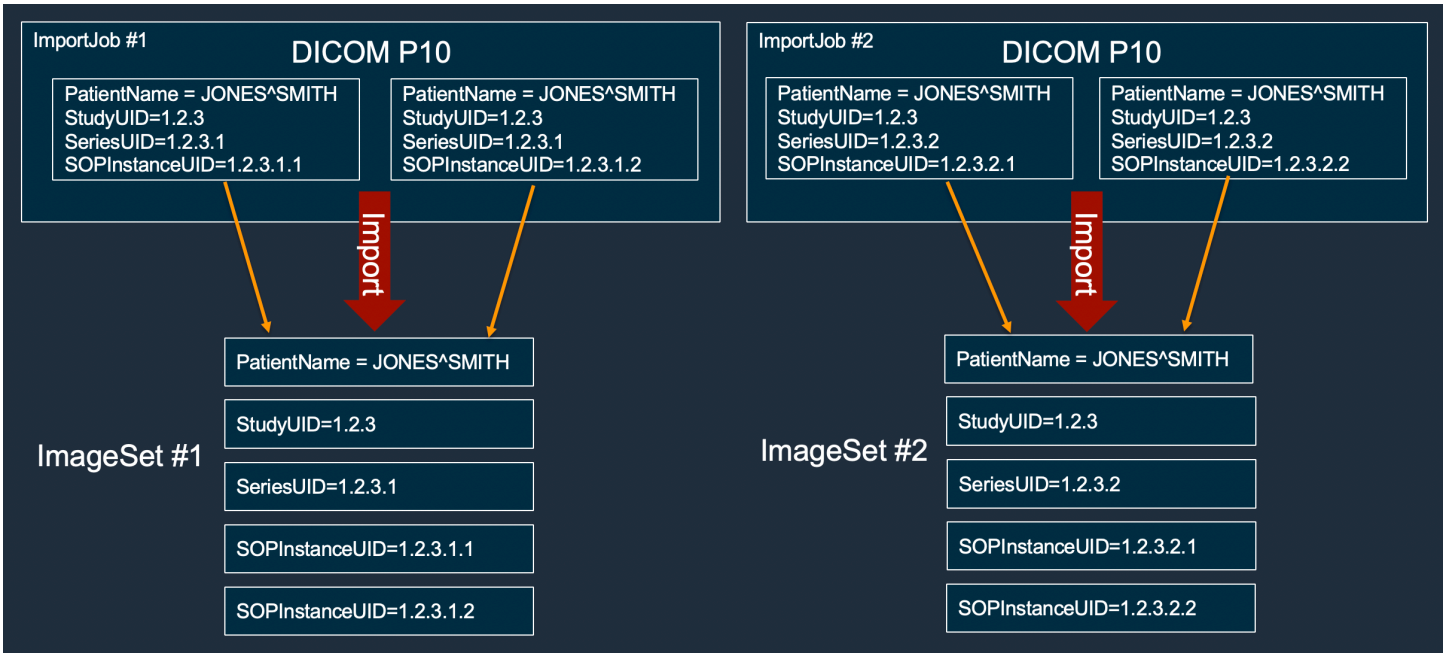
El siguiente ejemplo muestra la estructura de los metadatos del conjunto de imágenes en formato JSON.


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

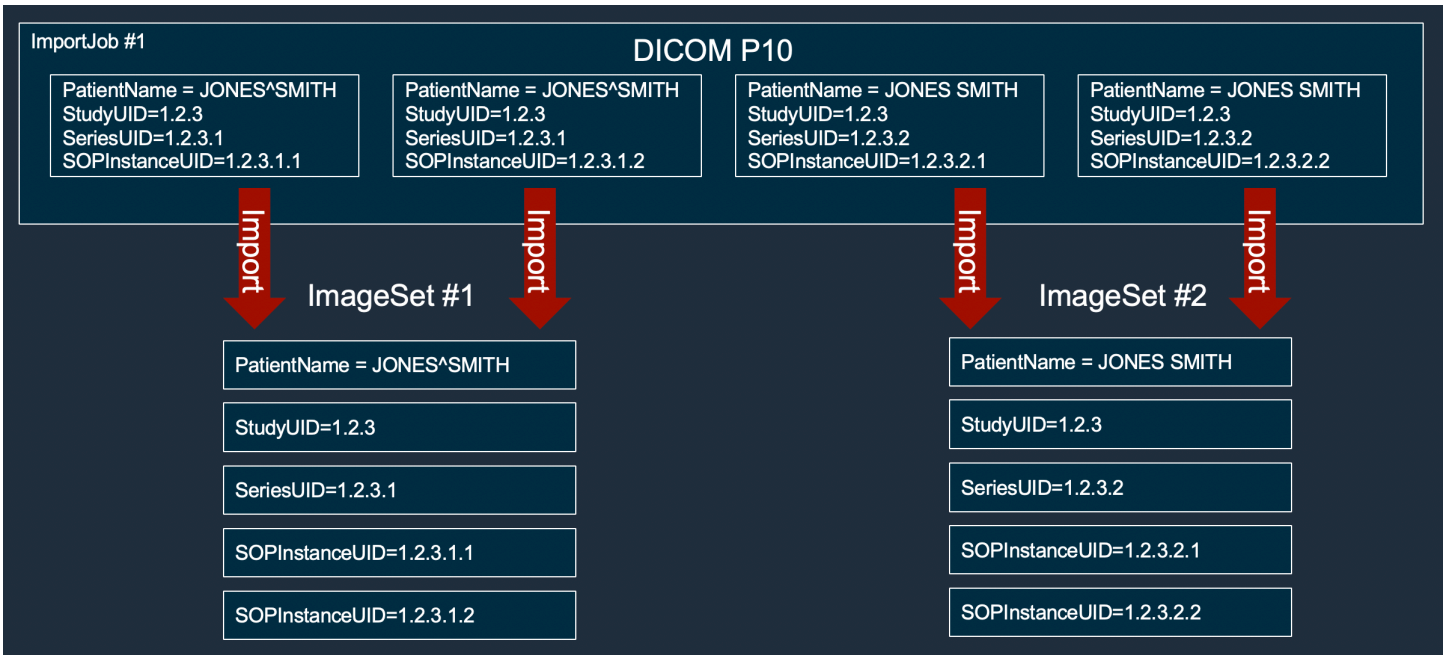
Ejemplo de creación de un conjunto de imágenes: varios trabajos de importación

El ejemplo siguiente muestra cómo varios trabajos de importación crean siempre nuevos conjuntos de imágenes y nunca los agregan a los existentes.



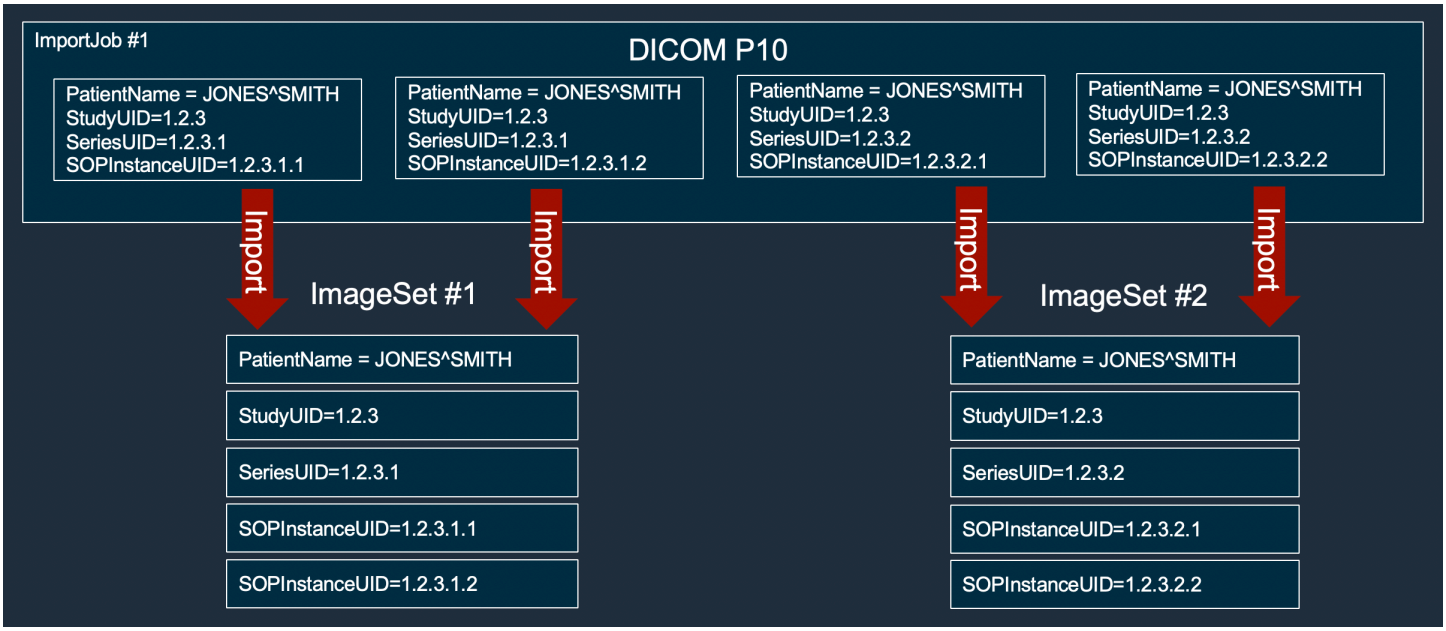
Ejemplo de creación de un conjunto de imágenes: trabajo de importación único con dos variantes

El ejemplo siguiente muestra un único trabajo de importación que crea dos conjuntos de imágenes, dado que las instancias 1 y 2 tienen nombres de pacientes diferentes a los de las instancias 3 y 4.



Ejemplo de creación de un conjunto de imágenes: trabajo de importación único con optimización

El ejemplo siguiente muestra un único trabajo de importación que crea dos conjuntos de imágenes para mejorar el rendimiento, aunque los nombres de los pacientes coincidan.



Búsqueda de conjuntos de imágenes

Utilice la `SearchImageSets` acción para ejecutar consultas de búsqueda en todos los conjuntos de imágenes de un almacén de `ACTIVE` HealthImaging datos. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `SearchImageSets`la AWS HealthImaging APIReferencia.

> **Note**
>
> Tenga en cuenta los siguientes puntos al buscar conjuntos de imágenes.
>
> - `SearchImageSets` acepta un parámetro de consulta de búsqueda único y devuelve una respuesta paginada de todos los conjuntos de imágenes que cumplen los criterios de coincidencia. Todas las consultas de intervalo de fechas se deben introducir como`(lowerBound, upperBound)`.
> - De forma predeterminada, `SearchImageSets` utiliza el `updatedAt` campo para ordenar en orden decreciente, del más reciente al más antiguo.

- Si creó su almacén de datos con un propietario del cliente AWS KMS clave, debe actualizar su AWS KMS política clave antes de interactuar con conjuntos de imágenes. Para más información, consulte Creación de claves administradas por el cliente.

Para buscar conjuntos de imágenes

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

> **Note**
>
> Los siguientes procedimientos muestran cómo buscar conjuntos de imágenes mediante los filtros `Series Instance UID` y de `Updated at` propiedades.

Series Instance UID

Busque conjuntos de imágenes mediante el filtro **`Series Instance UID`** de propiedades

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de detalles del almacén de datos y, por defecto, se seleccionará la pestaña Conjuntos de imágenes.
3. Elija el menú de filtros de propiedades y seleccione`Series Instance UID`.
4. En el campo Introduzca un valor para buscar, introduzca (pegue) la instancia UID de serie que le interese.

   > **Note**
   >
   > UIDLos valores de las instancias de serie deben ser idénticos a los que figuran en el Registro de identificadores DICOM únicos (UIDs). Tenga en cuenta que los requisitos incluyen una serie de números que contengan al menos un punto entre ellos. No se permiten puntos al principio o al final de la instancia de la serieUIDs. No se permiten letras ni espacios en blanco, así que tenga cuidado al copiar y pegarUIDs.

5. Seleccione el menú Intervalo de fechas, seleccione un intervalo de fechas para la instancia UID de la serie y seleccione Aplicar.
6. Elija Buscar.

   Las instancias de la serie UIDs que se encuentran dentro del intervalo de fechas seleccionado se devuelven en el orden más reciente de forma predeterminada.

Updated at

Busque conjuntos de imágenes mediante el filtro **`Updated at`** de propiedades

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de detalles del almacén de datos y, por defecto, se seleccionará la pestaña Conjuntos de imágenes.
3. Elija el menú de filtros de propiedades y elija`Updated at`.
4. Elija el menú Intervalo de fechas, seleccione un intervalo de fechas establecido para la imagen y elija Aplicar.
5. Elija Buscar.

   De forma predeterminada, los conjuntos de imágenes que se encuentran dentro del intervalo de fechas seleccionado se muestran en el orden más reciente.

AWS CLI y SDKs

C++

SDKpara C++

La función de utilidad para buscar conjuntos de imágenes.

```
//! 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 #1: EQUAL operador.

```
       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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
        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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
       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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
       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 API obtener más información, consulte SearchImageSetsen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Ejemplo 1: Para buscar conjuntos de imágenes con un EQUAL operador

El siguiente ejemplo `search-image-sets` de código utiliza el EQUAL operador para buscar conjuntos de imágenes en función de un valor específico.

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

Contenido de `search-criteria.json`

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

Salida:

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

Ejemplo 2: Para buscar conjuntos de imágenes con un BETWEEN operador, utilice DICOMStudyDate y DICOMStudyTime

El siguiente ejemplo de `search-image-sets` código busca conjuntos de imágenes con DICOM estudios generados entre el 1 de enero de 1990 (12:00 a. m.) y el 1 de enero de 2023 (12:00 a. m.).

Nota: DICOMStudyTime es opcional. Si no está presente, el valor de hora de las fechas indicado para el filtrado es a las 00:00 h (inicio del día).

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

Contenido de `search-criteria.json`

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

Salida:

```
{
```

```
    "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"
    }]
}
```

Ejemplo 3: Para buscar conjuntos de imágenes con un BETWEEN operador utilizando createdAt (previamente se conservaban los estudios de tiempo)

El siguiente ejemplo de `search-image-sets` código busca conjuntos de imágenes cuyos DICOM estudios persistan HealthImaging entre los rangos de tiempo de la zona horariaUTC.

Nota: Introdúzcalo createdAt en un formato de ejemplo («1985-04-12T 23:20:50.52 Z»).

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

Contenido de `search-criteria.json`

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

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

Salida:

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

Ejemplo 4: Para buscar conjuntos de imágenes con un EQUAL operador activado y activado DICOMSeriesInstanceUID y ordenar las respuestas por orden en el campo BETWEEN updatedAt ASC updatedAt

En el siguiente ejemplo de `search-image-sets` código se buscan conjuntos de imágenes con un EQUAL operador activado DICOMSeriesInstanceUID y BETWEEN activado updatedAt y se ordenan las ASC respuestas por updatedAt campo.

Nota: Escríbelo updatedAt en un formato de ejemplo («1985-04-12T 23:20:50 .52 Z»).

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

Contenido 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"
    }
}
```

Salida:

```
{
    "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 obtener más información, consulte Búsqueda de conjuntos de imágenes en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte SearchImageSetsen AWS CLI Referencia de comandos.

Java

SDKpara Java 2.x

La función de utilidad para buscar conjuntos de imágenes.

```
   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 #1: EQUAL operador.

```
        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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
        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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
        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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
        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 API obtener más información, consulte SearchImageSetsen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

La función de utilidad para buscar conjuntos de imágenes.

```
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 #1: EQUAL operador.

```
    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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
    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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
    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 API obtener más información, consulte SearchImageSetsen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

La función de utilidad para buscar conjuntos de imágenes.

```
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 #1: EQUAL operador.

```
        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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
        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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
        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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
        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}")
```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener API más información, consulte en SearchImageSetsAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Obtención de las propiedades del conjunto de imágenes

Utilice la `GetImageSet` acción para devolver las propiedades de un conjunto de imágenes determinado HealthImaging. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `GetImageSet`la AWS HealthImaging APIReferencia.

> **Note**
>
> De forma predeterminada, AWS HealthImaging devuelve las propiedades de la última versión de un conjunto de imágenes. Para ver las propiedades de una versión anterior de un conjunto de imágenes, indique la `versionId` al realizar la solicitud.
> Utilice `GetDICOMInstance` HealthImaging la representación de un DICOMweb servicio para devolver un binario de DICOM instancia (`.dcm`archivo). Para obtener más información, consulte Obtener una DICOM instancia de HealthImaging.

Cómo obtener las propiedades de un conjunto de imágenes

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de detalles del almacén de datos y, por defecto, se seleccionará la pestaña Conjuntos de imágenes.
3. Seleccione un conjunto de imágenes.

   Se abrirá la página de detalles del conjunto de imágenes, que muestra las propiedades del conjunto de imágenes.

AWS CLI y SDKs

CLI

AWS CLI

Obtención de las propiedades de un conjunto de imágenes

En el siguiente ejemplo de código `get-image-set` se obtienen las propiedades de un conjunto de imágenes.

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

Salida:

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

```
}
```

Para obtener más información, consulte Obtener las propiedades del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetImageSeten AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetImageSeten AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

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

```
  //     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 API obtener más información, consulte GetImageSeten AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en GetImageSetAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Obtención de metadatos de conjuntos de imágenes

Utilice la `GetImageSetMetadata` acción para recuperar los metadatos de un conjunto de imágenes determinado HealthImaging. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `GetImageSetMetadata`la AWS HealthImaging APIReferencia.

> **Note**
>
> De forma predeterminada, HealthImaging devuelve los atributos de metadatos de la última versión de un conjunto de imágenes. Para ver los metadatos de una versión anterior de un conjunto de imágenes, indique la `versionId` al realizar la solicitud.
> Los metadatos del conjunto de imágenes se comprimen con un objeto `gzip` y se devuelven como un JSON objeto. Por lo tanto, debe descomprimir el JSON objeto antes de ver los metadatos normalizados. Para obtener más información, consulte Normalización de metadatos.
> Utilice `GetDICOMInstanceMetadata` HealthImaging la representación de un DICOMweb servicio para devolver los metadatos de la DICOM instancia (`.json`archivo). Para obtener más información, consulte Obtener metadatos de DICOM instancias de HealthImaging.

Cómo obtener metadatos de conjuntos de imágenes

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de detalles del almacén de datos y, por defecto, se seleccionará la pestaña Conjuntos de imágenes.
3. Seleccione un conjunto de imágenes.

   Se abrirá la página de Detalles del conjunto de imágenes, y los metadatos del conjunto de imágenes aparecerán en la sección del Visor de metadatos de conjuntos de imágenes.

AWS CLI y SDKs

C++

SDKpara C++

Función de utilidad para obtener metadatos del conjunto de imágenes.

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

Obtener metadatos del conjunto de imágenes sin versión.

```
       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;
       }
```

Obtener metadatos del conjunto de imágenes con la versión.

```
       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 API obtener más información, consulte GetImageSetMetadataen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Ejemplo 1: obtención de los metadatos de un conjunto de imágenes sin versión

En el siguiente ejemplo de código `get-image-set-metadata` se obtienen los metadatos de un conjunto de imágenes sin especificar una versión.

Nota: El parámetro `outfile` es obligatorio

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

Los metadatos devueltos se comprimen con gzip y se almacenan en el archivo studymetadata.json.gz. Para ver el contenido del JSON objeto devuelto, primero debe descomprimirlo.

Salida:

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

Ejemplo 2: obtención de los metadatos de un conjunto de imágenes con versión

En el siguiente ejemplo de código `get-image-set-metadata` se obtienen los metadatos de un conjunto de imágenes con una versión especificada.

Nota: El parámetro `outfile` es obligatorio

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

Los metadatos devueltos se comprimen con gzip y se almacenan en el archivo studymetadata.json.gz. Para ver el contenido del JSON objeto devuelto, primero debe descomprimirlo.

Salida:

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

Para obtener más información, consulte Obtener los metadatos del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetImageSetMetadataen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetImageSetMetadataen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

Función de utilidad para obtener metadatos del conjunto de imágenes.

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

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

Obtener metadatos del conjunto de imágenes sin versión.

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

Obtener metadatos del conjunto de imágenes con la versión.

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
```

```
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- Para API obtener más información, consulte GetImageSetMetadataen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

Función de utilidad para obtener metadatos del conjunto de imágenes.

```
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
```

Obtener metadatos del conjunto de imágenes sin versión.

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

Obtener metadatos del conjunto de imágenes con la versión.

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

```
                datastoreId=datastore_id,
                versionId=version_id,
            )
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en GetImageSetMetadataAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Obtención de datos de píxeles de los conjuntos de imágenes

Los conjuntos de imágenes son la información de los píxeles existentes en un conjunto de imágenes y que forman una imagen médica en 2D. Utilice la `GetImageFrame` acción para recuperar un marco de imagen HTJ2K codificado para un conjunto de imágenes determinado. HealthImaging Los menús siguientes proporcionan ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `GetImageFrame`la AWS HealthImaging APIReferencia.

> **Note**
>
> Durante la importación, AWS HealthImaging codifica todos los fotogramas de imagen en un formato HTJ2K sin pérdidas, por lo que deben decodificarse antes de visualizarlos en un visor de imágenes. Para obtener más información, consulte Bibliotecas de decodificación HTJ2K.
> Utilice`GetDICOMInstanceFrames`, una representación HealthImaging de un DICOMweb servicio, para devolver los marcos de las DICOM instancias (solicitud). `multipart` Para obtener más información, consulte Obtener marcos de DICOM instancia de HealthImaging.

Obtención de datos de píxeles de un conjunto de imágenes

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

> **Note**
>
> Se debe acceder a los marcos de imagen y decodificarlos mediante programación, ya que no hay ningún visor de imágenes disponible en el AWS Management Console.
> Para más información sobre la decodificación y la visualización de marcos de imágenes, consulte Bibliotecas de decodificación HTJ2K.

AWS CLI y SDKs

C++

SDKpara 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 API obtener más información, consulte GetImageFrameen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Obtención de datos de píxeles de un conjunto de imágenes

En el siguiente ejemplo de código `get-image-frame` se obtiene un marco de una imagen.

```
aws medical-imaging get-image-frame \
```

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

Nota: Este ejemplo de código no incluye la salida porque la GetImageFrame acción devuelve un flujo de datos de píxeles al archivo imageframe.jph. Para obtener información sobre la decodificación y la visualización de marcos de imágenes, consulte las bibliotecas de decodificación. HTJ2K

Para obtener más información, consulte Obtener los datos de píxeles del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetImageFrameen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetImageFrameen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte GetImageFrameen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en GetImageFrameAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Modificación de conjuntos de imágenes con AWS HealthImaging

DICOMLos trabajos de importación suelen requerir que modifique los conjuntos de imágenes por los siguientes motivos:

- Seguridad del paciente
- Coherencia de datos
- Reducción de los costos de almacenamiento

> **Importante**
>
> Durante la importación, HealthImaging procesa los binarios de DICOM instancia (`.dcm`archivos) y los transforma en conjuntos de imágenes. Utilice las acciones nativas de HealthImaging la nube (APIs) para gestionar los almacenes de datos y los conjuntos de imágenes. Utilice HealthImaging la representación de los DICOMweb servicios para devolver DICOMweb las respuestas.

HealthImaging proporciona varios componentes nativos de la nube APIs para simplificar el proceso de modificación del conjunto de imágenes. En los temas siguientes se describe cómo modificar los conjuntos de imágenes mediante AWS CLI y AWS SDKs.

Temas

Listado de versiones de conjuntos de imágenes

Utilice la `ListImageSetVersions` acción para mostrar el historial de versiones de un conjunto de imágenes HealthImaging. Los menús siguientes proporcionan un procedimiento para AWS

Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `ListImageSetVersions`la AWS HealthImaging APIReferencia.

> **Note**
>
> AWS HealthImaging registra todos los cambios realizados en un conjunto de imágenes. Al actualizar los metadatos de un conjunto de imágenes, se crea una nueva versión en el historial de conjuntos de imágenes. Para obtener más información, consulte Actualización de los metadatos de un conjunto de imágenes.

Cómo enumerar las versiones de los conjuntos de imágenes

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de detalles del almacén de datos y, por defecto, se seleccionará la pestaña Conjuntos de imágenes.
3. Seleccione un conjunto de imágenes.

   Se abrirá la página de Detalles del conjunto de imágenes.

   La versión del conjunto de imágenes aparece en la sección de Detalles del conjunto de imágenes.

AWS CLI y SDKs

CLI

AWS CLI

Enumeración de las versiones de un conjunto de imágenes

En el siguiente ejemplo de código `list-image-set-versions` se enumera el historial de versiones de un conjunto de imágenes.

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

Salida:


```
{
    "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 obtener más información, consulte Cómo enumerar las versiones de conjuntos de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte ListImageSetVersionsen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte ListImageSetVersionsen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

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

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

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

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
```

```
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Para API obtener más información, consulte ListImageSetVersionsen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en ListImageSetVersionsAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

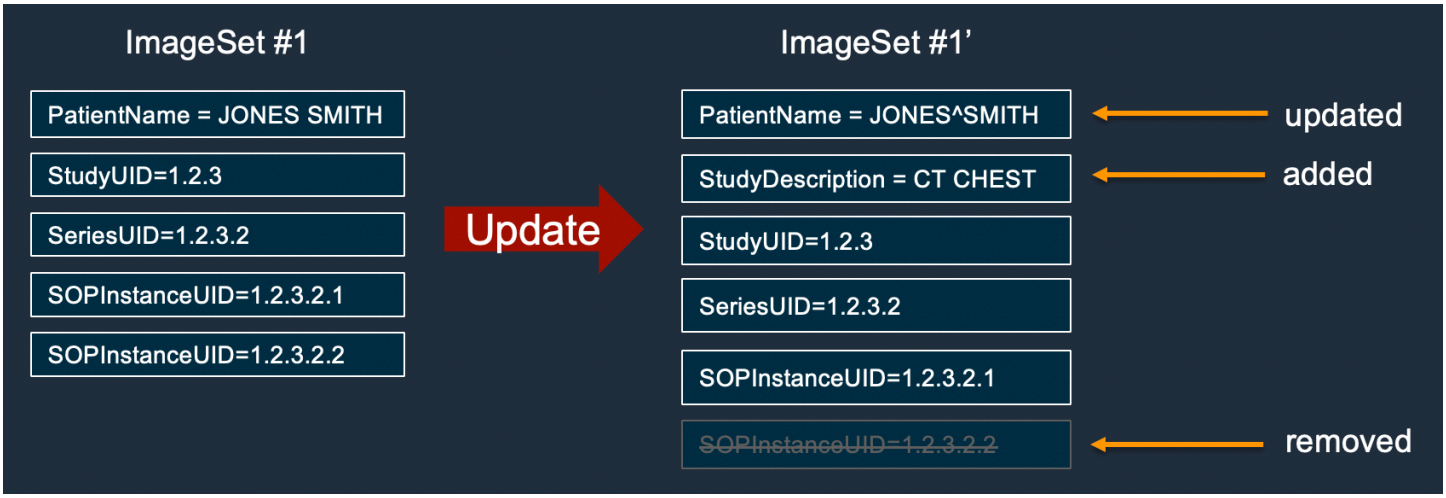
Actualización de los metadatos de un conjunto de imágenes

Utilice la `UpdateImageSetMetadata` acción para actualizar los metadatos del conjunto de imágenes en AWS HealthImaging. Puede utilizar este proceso asíncrono para añadir, actualizar y eliminar los atributos de metadatos del conjunto de imágenes, que son manifestaciones de los elementos de DICOM normalización que se crean durante la importación. Con esta `UpdateImageSetMetadata` acción, también puede eliminar series e SOP instancias para mantener los conjuntos de imágenes sincronizados con los sistemas externos y desidentificar los metadatos de los conjuntos de imágenes. Para obtener más información, consulte `UpdateImageSetMetadata`la AWS HealthImaging APIReferencia.

> **Note**
>
> DICOMLas importaciones reales requieren actualizar, añadir y eliminar atributos de los metadatos del conjunto de imágenes. Tenga en cuenta los siguientes puntos al actualizar los metadatos del conjunto de imágenes:
>
> - Al actualizar los metadatos de un conjunto de imágenes, se crea una nueva versión en el historial de conjuntos de imágenes. Para obtener más información, consulte Listado de versiones de conjuntos de imágenes. Para volver a un ID de versión anterior del conjunto de imágenes, utilice el `revertToVersionId`parámetro opcional.
> - La actualización de los metadatos de los conjuntos de imágenes es un proceso asíncrono. Por lo `imageSetState`tanto, hay elementos de `imageSetWorkflowStatus`respuesta disponibles para proporcionar el estado y el estado respectivos de un conjunto de imágenes que se está actualizando. No puede realizar otras operaciones de escritura en un conjunto `LOCKED` de imágenes.
> - Si la `UpdateImageSetMetadata` acción no se realiza correctamente, llame y revise el elemento de `message`respuesta para comprobarlo `common errors.`
> - DICOMlas restricciones de elementos se aplican a las actualizaciones de metadatos. El parámetro de `force`solicitud le permite actualizar los elementos en los casos en los que desee anularlosDICOMrestricciones de metadatos.
> - Defina el parámetro de `force`solicitud para forzar la finalización de la `UpdateImageSetMetadata` acción. Si se establece este parámetro, se permiten las siguientes actualizaciones en un conjunto de imágenes:
>   - Actualización de `Tag.StudyInstanceUID` los `Tag.StudyID` atributos `Tag.SeriesInstanceUIDTag.SOPInstanceUID`, y

- Añadir, eliminar o actualizar elementos de DICOM datos privados a nivel de instancia

El siguiente diagrama representa los metadatos del conjunto de imágenes en el que se están actualizando HealthImaging.



Cómo actualizar los metadatos de un conjunto de imágenes

Elija una pestaña en función de sus preferencias de acceso a AWS HealthImaging.

AWS CLI y SDKs

CLI

AWS CLI

Ejemplo 1: Para insertar o actualizar un atributo en los metadatos del conjunto de imágenes

En el siguiente `update-image-set-metadata` ejemplo, se inserta o actualiza un atributo en los metadatos del conjunto de imágenes.

```
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
```

Contenido de `metadata-updates.json`

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

Salida:

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

Ejemplo 2: Para eliminar un atributo de los metadatos del conjunto de imágenes

En el siguiente `update-image-set-metadata` ejemplo, se elimina un atributo de los metadatos del conjunto de imágenes.

```
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
```

Contenido de `metadata-updates.json`

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

Salida:

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

Ejemplo 3: Para eliminar una instancia de los metadatos del conjunto de imágenes

En el siguiente `update-image-set-metadata` ejemplo, se elimina una instancia de los metadatos del conjunto de imágenes.

```
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
```

Contenido 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\": {}}}}}}"
    }
}
```

Salida:

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

```
}
```

Ejemplo 4: Para revertir un conjunto de imágenes a una versión anterior

El siguiente `update-image-set-metadata` ejemplo muestra cómo revertir un conjunto de imágenes a una versión anterior. CopyImageSet y UpdateImageSetMetadata las acciones crean nuevas versiones de conjuntos de imágenes.

```
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"}'
```

Salida:

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

Ejemplo 5: Para añadir un elemento de DICOM datos privados a una instancia

En el siguiente `update-image-set-metadata` ejemplo, se muestra cómo añadir un elemento privado a una instancia específica dentro de un conjunto de imágenes. El DICOM estándar permite que los elementos de datos privados comuniquen información que no puede estar contenida en los elementos de datos estándar. Puede crear, actualizar y eliminar elementos de datos privados con esta UpdateImageSetMetadata acción.

```
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
```

Contenido 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\"}}}}}}"
    }
}
```

Salida:

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

Ejemplo 6: Para actualizar un elemento de DICOM datos privados en una instancia

El siguiente `update-image-set-metadata` ejemplo muestra cómo actualizar el valor de un elemento de datos privado que pertenece a una instancia dentro de un conjunto de imágenes.

```
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
```

Contenido 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\"}}}}}}"
    }
}
```

Salida:

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

Ejemplo 7: Para actualizar a SOPInstanceUID con el parámetro force

En el siguiente `update-image-set-metadata` ejemploSOPInstanceUID, se muestra cómo actualizar a mediante el parámetro force para anular las restricciones de los DICOM metadatos.

```
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
```

Contenido 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\"}}}}}}}"
    }
}
```

Salida:

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

Para obtener más información, consulte Actualización de los metadatos del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte UpdateImageSetMetadataen AWS CLI Referencia de comandos.

Java

SDKpara 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: insertar o actualizar un 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: eliminar un 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: eliminar una instancia.

```
                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: volver a una versión 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 API obtener más información, consulte UpdateImageSetMetadataen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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: inserte o actualice un atributo y fuerce la actualización.

```
        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: eliminar un 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: eliminar una instancia.

```
        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: volver a una versión anterior.

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

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

- Para API obtener más información, consulte UpdateImageSetMetadataen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

Caso de uso #1: inserta o actualiza un 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: eliminar un 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: eliminar una instancia.

```
        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: volver a una versión anterior.

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

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

- Para API obtener más información, consulte UpdateImageSetMetadataen AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Copia de conjuntos de imágenes

Usa la `CopyImageSet` acción para copiar un conjunto de imágenes HealthImaging. Este proceso asíncrono se utiliza para copiar el contenido de un conjunto de imágenes en un conjunto de imágenes nuevo o existente. Puede copiar en un conjunto de imágenes nuevo para dividir un conjunto de imágenes, así como para crear una copia independiente. También puede copiar en un conjunto de imágenes existente para combinar dos conjuntos de imágenes. Para obtener más información, consulte `CopyImageSet`la AWS HealthImaging APIReferencia.

> **Note**
>
> Tenga en cuenta los siguientes puntos al utilizar la `CopyImageSet` acción:

- La `CopyImageSet` acción creará un nuevo conjunto de imágenes o una nueva versión del`destinationImageSet`. Para obtener más información, consulte Listado de versiones de conjuntos de imágenes.
- La copia es un proceso asíncrono. Por lo tanto, los elementos de respuesta de estado (`imageSetStateimageSetWorkflowStatus`) y estado () están disponibles para indicarle qué operación se está realizando en un conjunto de imágenes bloqueado. No se pueden realizar otras operaciones de escritura en un conjunto de imágenes bloqueado.
- `CopyImageSet`requiere que la SOP instancia UIDs sea única dentro de un conjunto de imágenes.
- Puede copiar subconjuntos de SOP instancias utilizando `copiableAttributes`. Esto le permite seleccionar una o más SOP instancias de las `sourceImageSet` que desea copiar a las`destinationImageSet`.
- Si la `CopyImageSet` acción no se realiza correctamente, llame `GetImageSet` y revise la `message`propiedad. Para obtener más información, consulte Obtención de las propiedades del conjunto de imágenes.
- DICOMLas importaciones reales pueden dar como resultado varios conjuntos de imágenes por DICOM serie. La `CopyImageSet` acción requiere `sourceImageSet` y debe tener metadatos consistentes, `destinationImageSet` a menos que se suministre el `force`parámetro opcional.
- Defina el `force`parámetro para forzar la operación, incluso si hay elementos de metadatos incoherentes entre el `sourceImageSet` y`destinationImageSet`. En estos casos, los metadatos del paciente, del estudio y de la serie permanecen inalterados en el`destinationImageSet`.

Cómo copiar conjuntos de imágenes

Elija una pestaña en función de sus preferencias de acceso a AWS HealthImaging.

AWS CLI y SDKs

CLI

AWS CLI

Ejemplo 1: copia de un conjunto de imágenes sin un destino.

El siguiente `copy-image-set` ejemplo hace una copia duplicada de un conjunto de imágenes sin destino.

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

Salida:

```
{
    "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"
}
```

Ejemplo 2: copia de un conjunto de imágenes con un destino.

El siguiente `copy-image-set` ejemplo hace una copia duplicada de un conjunto de imágenes con un 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"} }'
```

Salida:

```
{
    "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"
}
```

Ejemplo 3: Para copiar un subconjunto de instancias de un conjunto de imágenes de origen a un conjunto de imágenes de destino.

En el siguiente `copy-image-set` ejemplo, se copia una DICOM instancia del conjunto de imágenes de origen al conjunto de imágenes de destino. El parámetro de fuerza se proporciona para anular las inconsistencias en los atributos de los niveles paciente, estudio y serie.

```
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
```

Salida:

```
{
    "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 obtener más información, consulte Copiar un conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte CopyImageSeten AWS CLI Referencia de comandos.

Java

SDKpara 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;
        }
    }
```

Función de utilidad para crear atributos copiables.

```
    /**
     * 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 API obtener más información, consulte CopyImageSeten AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

Función de utilidad para copiar un conjunto de imágenes.

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

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

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

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

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

        params.copyImageSetInformation.dicomCopies = copySubsetsJson;
    }

    const response = await medicalImagingClient.send(
        new CopyImageSetCommand(params)
    );
    console.log(response);
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //      datastoreId: '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 un conjunto de imágenes sin destino.

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

Copie un conjunto de imágenes con un destino.

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

Copie un subconjunto de un conjunto de imágenes con un destino y fuerce la copia.

```
    await copyImageSet(
        "12345678901234567890123456789012",
```

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

- Para API obtener más información, consulte CopyImageSeten AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

Función de utilidad para copiar un conjunto de imágenes.

```
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 un conjunto de imágenes sin 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,
            )
```

Copie un conjunto de imágenes con un 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,
            )
```

Copia un subconjunto de un conjunto de imágenes.

```
            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,
            )
```

El código siguiente crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en CopyImageSetAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Eliminación de un conjunto de imágenes

Usa la `DeleteImageSet` acción para eliminar un conjunto de imágenes HealthImaging. Los menús siguientes proporcionan un procedimiento para AWS Management Console y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte `DeleteImageSet`la AWS HealthImaging APIReferencia.

Cómo eliminar conjuntos de imágenes

Elija un menú según sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de Detalles del almacén de datos y, por defecto, se seleccionará la pestaña Conjuntos de imágenes.
3. Elija un conjunto de imágenes y, luego, Eliminar.

   Se abrirá la ventana emergente Eliminar conjunto de imágenes.
4. Indique el ID del conjunto de imágenes y elija Eliminar conjunto de imágenes.

AWS CLI y SDKs

C++

SDKpara 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 API obtener más información, consulte DeleteImageSeten AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Eliminación de un conjunto de imágenes

En el siguiente ejemplo de código `delete-image-set` se elimina un conjunto de imágenes.

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

Salida:

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

Para obtener más información, consulte Eliminar un conjunto de imágenes en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte DeleteImageSeten AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte DeleteImageSeten AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte DeleteImageSeten AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en DeleteImageSetAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Etiquetar recursos con AWS HealthImaging

Puede asignar metadatos a HealthImaging los recursos (almacenes de datos y conjuntos de imágenes) en forma de etiquetas. Cada etiqueta es una marca que consta de una clave y un valor definidos por el usuario. Las etiquetas son útiles a la hora de administrar, identificar, organizar, buscar y filtrar recursos.

> **Importante**
>
> No almacene información de salud protegida (PHI), información de identificación personal (PII) u otra información confidencial o delicada en etiquetas. Las etiquetas no se han diseñado para usarse con información privada o confidencial.

En los siguientes temas se describe cómo utilizar las operaciones de HealthImaging etiquetado mediante las letras AWS Management Console AWS CLI, y AWS SDKs. Para obtener más información, consulte Etiquetar AWS los recursos en la Referencia general de AWS Guía.

Temas

- Etiquetado de un recurso
- Listado de etiquetas de un recurso
- Eliminación de las etiquetas de un recurso

Etiquetado de un recurso

Usa la `TagResource`acción para etiquetar almacenes de datos y conjuntos de imágenes AWS HealthImaging. En los siguientes ejemplos de código se describe cómo utilizar la `TagResource` acción con AWS Management Console, AWS CLI, y AWS SDKs. Para obtener más información, consulte Etiquetar su AWS recursos en el Referencia general de AWS Guía.

Para etiquetar un recurso

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.

2. Elija un almacén de datos.

   Se abrirá la página de Detalles del almacén de datos.
3. Elija la pestaña Detalles.
4. En la sección Etiquetas, elija Administrar etiquetas.

   Se abrirá la página de Administrar etiquetas.
5. Elija Añadir nueva etiqueta.
6. Ingrese una clave y un valor (opcional).
7. Elija Guardar cambios.

AWS CLI y SDKs

CLI

AWS CLI

Ejemplo 1: etiquetado de un almacén de datos

En los siguientes ejemplos de código `tag-resource` se etiqueta un almacén de datos.

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

Este comando no genera ninguna salida.

Ejemplo 2: etiquetado de un conjunto de imágenes

En los siguientes ejemplos de código `tag-resource` se etiqueta un conjunto de imágenes.

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

Este comando no genera ninguna salida.

Para obtener más información, consulte Etiquetar recursos con AWS HealthImagingen el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte TagResourceen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte TagResourceen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte TagResourceen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en TagResourceAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Listado de etiquetas de un recurso

Utilice la `ListTagsForResource`acción para enumerar las etiquetas de los almacenes de datos y los conjuntos de imágenes AWS HealthImaging. En los siguientes ejemplos de código se describe cómo utilizar la `ListTagsForResource` acción con el AWS Management Console, AWS CLI, y AWS SDKs. Para obtener más información, consulte Etiquetar su AWS recursos en el Referencia general de AWS Guía.

Para enumerar las etiquetas de un recurso

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de Detalles del almacén de datos.
3. Elija la pestaña Detalles.

   En la sección Etiquetas, se muestran todas las etiquetas del almacén de datos.

AWS CLI y SDKs

CLI

AWS CLI

Ejemplo 1: enumeración de las etiquetas de recursos de un almacén de datos

En el siguiente ejemplo de código `list-tags-for-resource` se enumeran las etiquetas de un almacén de datos.

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

Salida:

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

Ejemplo 2: enumeración de las etiquetas de recursos de un conjunto de imágenes

En el siguiente ejemplo de código `list-tags-for-resource` se enumeran las etiquetas de un conjunto de imágenes.

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

Salida:

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

Para obtener más información, consulte Etiquetar recursos con AWS HealthImagingen el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte ListTagsForResourceen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte ListTagsForResourceen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte ListTagsForResourceen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en ListTagsForResourceAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y ejecutarlo en el AWS Repositorio de ejemplos de código.

Eliminación de las etiquetas de un recurso

Utilice esta `UntagResource`acción para desetiquetar los almacenes de datos y los conjuntos de imágenes. AWS HealthImaging En los siguientes ejemplos de código se describe cómo utilizar la `UntagResource` acción con AWS Management Console, AWS CLI, y AWS SDKs. Para obtener más información, consulte Etiquetar su AWS recursos en el Referencia general de AWS Guía.

Para quitar la etiqueta de un recurso

Elija un menú en función de sus preferencias de acceso a AWS HealthImaging.

AWS Consola

1. Abra la página de almacenes de datos de la HealthImaging consola.
2. Elija un almacén de datos.

   Se abrirá la página de Detalles del almacén de datos.
3. Elija la pestaña Detalles.
4. En la sección Etiquetas, elija Administrar etiquetas.

   Se abrirá la página de Administrar etiquetas.
5. Elija Eliminar junto a la etiqueta que desea eliminar.
6. Elija Guardar cambios.

AWS CLI y SDKs

CLI

AWS CLI

Ejemplo 1: eliminación de las etiquetas de un almacén de datos

En el siguiente ejemplo de código `untag-resource` se eliminan las etiquetas de un almacén de datos.

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

Este comando no genera ninguna salida.

Ejemplo 2: eliminación de las etiquetas de un conjunto de imágenes

En el siguiente ejemplo de código `untag-resource` se eliminan las etiquetas de un conjunto de imágenes.

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

Este comando no genera ninguna salida.

Para obtener más información, consulte Etiquetar recursos con AWS HealthImagingen el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte UntagResourceen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte UntagResourceen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte UntagResourceen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en UntagResourceAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información sobre. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Ejemplos de código para HealthImaging usar AWS SDKs

Los siguientes ejemplos de código muestran cómo usarlo HealthImaging con un AWS kit de desarrollo de software (SDK).

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Los escenarios son ejemplos de código que muestran cómo realizar tareas específicas llamando a múltiples funciones dentro de un servicio o combinándolas con otras Servicios de AWS.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

Introducción

Hola HealthImaging

Los siguientes ejemplos de código muestran cómo empezar a usarlo HealthImaging.

C++

SDKpara C++

Código para el CMakeLists CMake archivo.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 del archivo de origen 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 API obtener más información, consulte ListDatastoresen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte ListDatastoresen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

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

logger = logging.getLogger(__name__)
```

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

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


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

- Para obtener API más información, consulte en ListDatastoresAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Ejemplos de código

- Ejemplos básicos de HealthImaging uso AWS SDKs
  - Hola HealthImaging
  - Acciones de HealthImaging uso AWS SDKs
    - CopyImageSetÚselo con un AWS SDKo CLI
    - CreateDatastoreÚselo con un AWS SDKo CLI
    - DeleteDatastoreÚselo con un AWS SDKo CLI
    - DeleteImageSetÚselo con un AWS SDKo CLI
    - GetDICOMImportJobÚselo con un AWS SDKo CLI
    - GetDatastoreÚselo con un AWS SDKo CLI
    - GetImageFrameÚselo con un AWS SDKo CLI
    - GetImageSetÚselo con un AWS SDKo CLI
    - GetImageSetMetadataÚselo con un AWS SDKo CLI
    - ListDICOMImportJobsÚselo con un AWS SDKo CLI
    - ListDatastoresÚselo con un AWS SDKo CLI
    - ListImageSetVersionsÚselo con un AWS SDKo CLI
    - ListTagsForResourceÚselo con un AWS SDKo CLI
    - SearchImageSetsÚselo con un AWS SDKo CLI
    - StartDICOMImportJobÚselo con un AWS SDKo CLI
    - TagResourceÚselo con un AWS SDKo CLI
    - UntagResourceÚselo con un AWS SDKo CLI
    - UpdateImageSetMetadataÚselo con un AWS SDKo CLI
- Escenarios de HealthImaging uso AWS SDKs
  - Comience con los conjuntos HealthImaging de imágenes y los marcos de imágenes con un AWS SDK
  - Etiquetar un banco HealthImaging de datos mediante un AWS SDK
  - Etiquetar un conjunto HealthImaging de imágenes mediante un AWS SDK

Ejemplos básicos de HealthImaging uso AWS SDKs

Los siguientes ejemplos de código muestran cómo utilizar los conceptos básicos de AWS HealthImaging with AWS SDKs.

Ejemplos

- Hola HealthImaging
- Acciones de HealthImaging uso AWS SDKs
  - CopyImageSetÚselo con un AWS SDKo CLI
  - CreateDatastoreÚselo con un AWS SDKo CLI
  - DeleteDatastoreÚselo con un AWS SDKo CLI
  - DeleteImageSetÚselo con un AWS SDKo CLI
  - GetDICOMImportJobÚselo con un AWS SDKo CLI
  - GetDatastoreÚselo con un AWS SDKo CLI
  - GetImageFrameÚselo con un AWS SDKo CLI
  - GetImageSetÚselo con un AWS SDKo CLI
  - GetImageSetMetadataÚselo con un AWS SDKo CLI
  - ListDICOMImportJobsÚselo con un AWS SDKo CLI
  - ListDatastoresÚselo con un AWS SDKo CLI
  - ListImageSetVersionsÚselo con un AWS SDKo CLI
  - ListTagsForResourceÚselo con un AWS SDKo CLI
  - SearchImageSetsÚselo con un AWS SDKo CLI
  - StartDICOMImportJobÚselo con un AWS SDKo CLI
  - TagResourceÚselo con un AWS SDKo CLI
  - UntagResourceÚselo con un AWS SDKo CLI
  - UpdateImageSetMetadataÚselo con un AWS SDKo CLI

Hola HealthImaging

Los siguientes ejemplos de código muestran cómo empezar a usarlo HealthImaging.

C++

SDKpara C++

Código para el CMakeLists CMake archivo.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 del archivo de origen 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 API obtener más información, consulte ListDatastoresen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte ListDatastoresen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

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

logger = logging.getLogger(__name__)


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

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

```
        raise


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

- Para obtener API más información, consulte en ListDatastoresAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

Acciones de HealthImaging uso AWS SDKs

Los siguientes ejemplos de código muestran cómo realizar HealthImaging acciones individuales con AWS SDKs. Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones para configurar y ejecutar el código.

Estos extractos se denominan HealthImaging API y son extractos de código de programas más grandes que deben ejecutarse en su contexto. Puede ver las acciones en su contexto en Escenarios de HealthImaging uso AWS SDKs .

Los siguientes ejemplos incluyen solo las acciones que se utilizan con mayor frecuencia. Para obtener una lista completa, consulte la AWS HealthImaging APIReferencia.

Ejemplos

- CopyImageSetÚselo con un AWS SDKo CLI
- CreateDatastoreÚselo con un AWS SDKo CLI
- DeleteDatastoreÚselo con un AWS SDKo CLI
- DeleteImageSetÚselo con un AWS SDKo CLI
- GetDICOMImportJobÚselo con un AWS SDKo CLI

- GetDatastoreÚselo con un AWS SDKo CLI
- GetImageFrameÚselo con un AWS SDKo CLI
- GetImageSetÚselo con un AWS SDKo CLI
- GetImageSetMetadataÚselo con un AWS SDKo CLI
- ListDICOMImportJobsÚselo con un AWS SDKo CLI
- ListDatastoresÚselo con un AWS SDKo CLI
- ListImageSetVersionsÚselo con un AWS SDKo CLI
- ListTagsForResourceÚselo con un AWS SDKo CLI
- SearchImageSetsÚselo con un AWS SDKo CLI
- StartDICOMImportJobÚselo con un AWS SDKo CLI
- TagResourceÚselo con un AWS SDKo CLI
- UntagResourceÚselo con un AWS SDKo CLI
- UpdateImageSetMetadataÚselo con un AWS SDKo CLI

CopyImageSetÚselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `CopyImageSet`.

CLI

AWS CLI

Ejemplo 1: copia de un conjunto de imágenes sin un destino.

El siguiente `copy-image-set` ejemplo hace una copia duplicada de un conjunto de imágenes sin destino.

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

Salida:

```
{
    "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"
}
```

Ejemplo 2: copia de un conjunto de imágenes con un destino.

El siguiente `copy-image-set` ejemplo hace una copia duplicada de un conjunto de imágenes con un 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"} }'
```

Salida:

```
{
    "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"
}
```

Ejemplo 3: Para copiar un subconjunto de instancias de un conjunto de imágenes de origen a un conjunto de imágenes de destino.

En el siguiente `copy-image-set` ejemplo, se copia una DICOM instancia del conjunto de imágenes de origen al conjunto de imágenes de destino. El parámetro de fuerza se proporciona para anular las inconsistencias en los atributos de los niveles paciente, estudio y serie.

```
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
```

Salida:

```
{
    "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 obtener más información, consulte Copiar un conjunto de imágenes en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte CopyImageSeten AWS CLI Referencia de comandos.

Java

SDKpara 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;
        }
    }
```

Función de utilidad para crear atributos copiables.

```
    /**
     * 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 API obtener más información, consulte CopyImageSeten AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

Función de utilidad para copiar un conjunto de imágenes.

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

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

```
    imageSetId = "xxxxxxxxxxxx",
    sourceVersionId = "1",
    destinationImageSetId = "",
    destinationVersionId = "",
    force = false,
    copySubsets = []
) => {
    try {
        const params = {
            datastoreId: datastoreId,
            sourceImageSetId: imageSetId,
            copyImageSetInformation: {
                sourceImageSet: {latestVersionId: sourceVersionId},
            },
            force: force
        };
        if (destinationImageSetId !== "" && destinationVersionId !== "") {
            params.copyImageSetInformation.destinationImageSet = {
                imageSetId: destinationImageSetId,
                latestVersionId: destinationVersionId,
            };
        }

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

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

            params.copyImageSetInformation.dicomCopies = copySubsetsJson;
        }
```

```
        const response = await medicalImagingClient.send(
            new CopyImageSetCommand(params)
        );
        console.log(response);
        // {
        //     '$metadata': {
        //         httpStatusCode: 200,
        //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
        //         extendedRequestId: undefined,
        //         cfId: undefined,
        //         attempts: 1,
        //         totalRetryDelay: 0
        //      },
        //       datastoreId: '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 un conjunto de imágenes sin destino.

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

Copie un conjunto de imágenes con un destino.

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

Copie un subconjunto de un conjunto de imágenes con un destino y fuerce la copia.

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

- Para API obtener más información, consulte CopyImageSeten AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

Función de utilidad para copiar un conjunto de imágenes.

```
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:
 ["1234567890123456789012345678901 2"].
        :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 un conjunto de imágenes sin 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,
            )
```

Copie un conjunto de imágenes con un 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,
            )
```

Copia un subconjunto de un conjunto de imágenes.

```
            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,
            )
```

El código siguiente crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en CopyImageSetAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`CreateDatastore`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `CreateDatastore`.

Bash

AWS CLI con el 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 API obtener más información, consulte CreateDatastoreen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Creación de un almacén de datos

En el siguiente ejemplo de código `create-datastore` se crea un almacén de datos con el nombre `my-datastore`.

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

Salida:

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

Para obtener más información, consulte Crear un banco de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte CreateDatastoreen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte CreateDatastoreen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte CreateDatastoreen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en CreateDatastoreAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`DeleteDatastore`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `DeleteDatastore`.

Bash

AWS CLI con el 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 API obtener más información, consulte DeleteDatastoreen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Eliminación de un almacén de datos

En el siguiente ejemplo de código `delete-datastore` se elimina un almacén de datos.

```
aws medical-imaging delete-datastore \
```

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

Salida:

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

Para obtener más información, consulte Eliminar un banco de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte DeleteDatastoreen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte DeleteDatastoreen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte DeleteDatastoreen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en DeleteDatastoreAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`DeleteImageSet`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `DeleteImageSet`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Introducción a los conjuntos y marcos de imágenes

C++

SDKpara 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 API obtener más información, consulte DeleteImageSeten AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Eliminación de un conjunto de imágenes

En el siguiente ejemplo de código `delete-image-set` se elimina un conjunto de imágenes.

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

Salida:

```
{
```

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

Para obtener más información, consulte Eliminar un conjunto de imágenes en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte DeleteImageSeten AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte DeleteImageSeten AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
```

```
  return response;
};
```

- Para API obtener más información, consulte DeleteImageSeten AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en DeleteImageSetAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`GetDICOMImportJob`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `GetDICOMImportJob`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Introducción a los conjuntos y marcos de imágenes

C++

SDKpara 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 API obtener más información, consulte G etDICOMImport Job en AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Obtención de las propiedades de un trabajo de importación DICOM

En el siguiente ejemplo de código `get-dicom-import-job` se obtienen las propiedades de un trabajo de importación DICOM.

```
aws medical-imaging get-dicom-import-job \
    --datastore-id "1234567890123456789012345678901 2" \
    --job-id "09876543210987654321098765432109"
```

Salida:

```
{
    "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 obtener más información, consulte Obtener las propiedades de los trabajos de importación en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte G etDICOMImport Job en AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte G etDICOMImport Job en AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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: ''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 API obtener más información, consulte G etDICOMImport Job en AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte G etDICOMImport Job en AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

GetDatastoreÚselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `GetDatastore`.

Bash

AWS CLI con el 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 API obtener más información, consulte GetDatastoreen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Obtención de las propiedades de un almacén de datos

En el siguiente ejemplo de código `get-datastore` se obtienen las propiedades de un almacén de datos.

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

Salida:

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

```
    }
}
```

Para obtener más información, consulte Obtener las propiedades del almacén de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetDatastoreen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetDatastoreen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte GetDatastoreen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

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

- Para obtener API más información, consulte en GetDatastoreAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`GetImageFrame`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `GetImageFrame`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Introducción a los conjuntos y marcos de imágenes

C++

SDKpara 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 API obtener más información, consulte GetImageFrameen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Obtención de datos de píxeles de un conjunto de imágenes

En el siguiente ejemplo de código `get-image-frame` se obtiene un marco de una imagen.

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

Nota: Este ejemplo de código no incluye la salida porque la GetImageFrame acción devuelve un flujo de datos de píxeles al archivo imageframe.jph. Para obtener información sobre la decodificación y la visualización de marcos de imágenes, consulte las bibliotecas de decodificación. HTJ2K

Para obtener más información, consulte Obtener datos de píxeles del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetImageFrameen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetImageFrameen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte GetImageFrameen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en GetImageFrameAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`GetImageSet`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `GetImageSet`.

CLI

AWS CLI

Obtención de las propiedades de un conjunto de imágenes

En el siguiente ejemplo de código `get-image-set` se obtienen las propiedades de un conjunto de imágenes.

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

Salida:

```
{
    "versionId": "1",
```

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

Para obtener más información, consulte Obtener las propiedades del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetImageSeten AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetImageSeten AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
```

```
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     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 API obtener más información, consulte GetImageSeten AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en GetImageSetAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`GetImageSetMetadata`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `GetImageSetMetadata`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Introducción a los conjuntos y marcos de imágenes

C++

SDKpara C++

Función de utilidad para obtener metadatos del conjunto de imágenes.

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

Obtener metadatos del conjunto de imágenes sin versión.

```
        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;
        }
```

Obtener metadatos del conjunto de imágenes con la versión.

```
        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 API obtener más información, consulte GetImageSetMetadataen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Ejemplo 1: obtención de los metadatos de un conjunto de imágenes sin versión

En el siguiente ejemplo de código `get-image-set-metadata` se obtienen los metadatos de un conjunto de imágenes sin especificar una versión.

Nota: El parámetro `outfile` es obligatorio

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

Los metadatos devueltos se comprimen con gzip y se almacenan en el archivo studymetadata.json.gz. Para ver el contenido del JSON objeto devuelto, primero debe descomprimirlo.

Salida:

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

Ejemplo 2: obtención de los metadatos de un conjunto de imágenes con versión

En el siguiente ejemplo de código `get-image-set-metadata` se obtienen los metadatos de un conjunto de imágenes con una versión especificada.

Nota: El parámetro `outfile` es obligatorio

```
aws medical-imaging get-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
```

```
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --version-id 1 \
    studymetadata.json.gz
```

Los metadatos devueltos se comprimen con gzip y se almacenan en el archivo studymetadata.json.gz. Para ver el contenido del JSON objeto devuelto, primero debe descomprimirlo.

Salida:

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

Para obtener más información, consulte Obtener los metadatos del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte GetImageSetMetadataen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte GetImageSetMetadataen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

Función de utilidad para obtener metadatos del conjunto de imágenes.

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

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

Obtener metadatos del conjunto de imágenes sin versión.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012"
    );
```

```
  } catch (err) {
    console.log("Error", err);
  }
```

Obtener metadatos del conjunto de imágenes con la versión.

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

- Para API obtener más información, consulte GetImageSetMetadataen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

Función de utilidad para obtener metadatos del conjunto de imágenes.

```
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
```

Obtener metadatos del conjunto de imágenes sin versión.

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

Obtener metadatos del conjunto de imágenes con la versión.

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

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en GetImageSetMetadataAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`ListDICOMImportJobs`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `ListDICOMImportJobs`.

CLI

AWS CLI

Enumeración de los trabajos de importación DICOM

En el siguiente ejemplo de código `list-dicom-import-jobs` se enumeran los trabajos de importación DICOM.

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

Salida:

```
{
    "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 obtener más información, consulte Listar los trabajos de importación en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte L istDICOMImport Jobs en AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte L istDICOMImport Jobs en AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/
dicom_import',
  //             datastoreId: '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 API obtener más información, consulte L istDICOMImport Jobs en AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte L Jobs en istDICOMImport AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`ListDatastores`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `ListDatastores`.

Bash

AWS CLI con el 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 API obtener más información, consulte ListDatastoresen AWS CLI Referencia de comandos.

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Enumeración de almacenes de datos

En el siguiente ejemplo de código `list-datastores` se enumeran los almacenes de datos disponibles.

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

Salida:

```
{
    "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 obtener más información, consulte Listar los almacenes de datos en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte ListDatastoresen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte ListDatastoresen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte ListDatastoresen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en ListDatastoresAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`ListImageSetVersions`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `ListImageSetVersions`.

CLI

AWS CLI

Enumeración de las versiones de un conjunto de imágenes

En el siguiente ejemplo de código `list-image-set-versions` se enumera el historial de versiones de un conjunto de imágenes.

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

Salida:

```
{
    "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 obtener más información, consulte Cómo enumerar las versiones de conjuntos de imágenes en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte ListImageSetVersionsen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte ListImageSetVersionsen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

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

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

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

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
```

```
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Para API obtener más información, consulte ListImageSetVersionsen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en ListImageSetVersionsAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`ListTagsForResource`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `ListTagsForResource`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Etiquetar un almacén de datos
- Etiquetar un conjunto de imágenes

CLI

AWS CLI

Ejemplo 1: enumeración de las etiquetas de recursos de un almacén de datos

En el siguiente ejemplo de código `list-tags-for-resource` se enumeran las etiquetas de un almacén de datos.

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

Salida:

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

Ejemplo 2: enumeración de las etiquetas de recursos de un conjunto de imágenes

En el siguiente ejemplo de código `list-tags-for-resource` se enumeran las etiquetas de un conjunto de imágenes.

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

Salida:

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

Para obtener más información, consulte Etiquetar recursos con AWS HealthImagingen el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte ListTagsForResourceen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte ListTagsForResourceen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte ListTagsForResourceen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en ListTagsForResourceAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`SearchImageSets`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `SearchImageSets`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Introducción a los conjuntos y marcos de imágenes

C++

SDKpara C++

La función de utilidad para buscar conjuntos de imágenes.

```
//! 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 #1: EQUAL operador.

```
       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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
        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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
        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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
        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 API obtener más información, consulte SearchImageSetsen AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Ejemplo 1: Para buscar conjuntos de imágenes con un EQUAL operador

El siguiente ejemplo `search-image-sets` de código utiliza el EQUAL operador para buscar conjuntos de imágenes en función de un valor específico.

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

Contenido de `search-criteria.json`

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

Salida:

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

Ejemplo 2: Para buscar conjuntos de imágenes con un BETWEEN operador, utilice DICOMStudyDate y DICOMStudyTime

El siguiente ejemplo de `search-image-sets` código busca conjuntos de imágenes con DICOM estudios generados entre el 1 de enero de 1990 (12:00 a. m.) y el 1 de enero de 2023 (12:00 a. m.).

Nota: DICOMStudyTime es opcional. Si no está presente, el valor de hora de las fechas indicado para el filtrado es a las 00:00 h (inicio del día).

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

Contenido de `search-criteria.json`

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

Salida:

```
{
```

```
    "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"
    }]
}
```

Ejemplo 3: Para buscar conjuntos de imágenes con un BETWEEN operador utilizando createdAt (previamente se conservaban los estudios de tiempo)

El siguiente ejemplo de `search-image-sets` código busca conjuntos de imágenes cuyos DICOM estudios persistan HealthImaging entre los rangos de tiempo de la zona horariaUTC.

Nota: Escríbalo createdAt en un formato de ejemplo («1985-04-12T 23:20:50.52 Z»).

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

Contenido de `search-criteria.json`

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

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

Salida:

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

Ejemplo 4: Para buscar conjuntos de imágenes con un EQUAL operador activado y activado DICOMSeriesInstanceUID y ordenar las respuestas por orden en el campo BETWEEN updatedAt ASC updatedAt

El siguiente ejemplo de `search-image-sets` código busca conjuntos de imágenes con un EQUAL operador activado DICOMSeriesInstanceUID y BETWEEN activado updatedAt y ordena las respuestas por ASC orden en el updatedAt campo.

Nota: Escríbalo updatedAt en un formato de ejemplo («1985-04-12T 23:20:50 .52 Z»).

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

Contenido 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"
    }
}
```

Salida:

```
{
    "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 obtener más información, consulte Búsqueda de conjuntos de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte SearchImageSetsen AWS CLI Referencia de comandos.

Java

SDKpara Java 2.x

La función de utilidad para buscar conjuntos de imágenes.

```
   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 #1: EQUAL operador.

```
        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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
        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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
        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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
        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 API obtener más información, consulte SearchImageSetsen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

La función de utilidad para buscar conjuntos de imágenes.

```
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 #1: EQUAL operador.

```
    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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
const datastoreId = "1234567890123456789012345678901 2";

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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
    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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
    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 API obtener más información, consulte SearchImageSetsen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

La función de utilidad para buscar conjuntos de imágenes.

```
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 #1: EQUAL operador.

```
        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: BETWEEN operador que usa DICOMStudyDate yDICOMStudyTime.

```
        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 #3: BETWEEN operador usandocreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
        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: el EQUAL operador se DICOMSeriesInstanceUID activa y BETWEEN updatedAt activa y ordena la respuesta en ASC orden en el updatedAt campo.

```
        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}")
```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener API más información, consulte en SearchImageSetsAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`StartDICOMImportJob`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `StartDICOMImportJob`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Introducción a los conjuntos y marcos de imágenes

C++

SDKpara 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 API obtener más información, consulte S tartDICOMImport Job en AWS SDK for C++ APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

CLI

AWS CLI

Inicio de un trabajo de importación DICOM

En el siguiente ejemplo de código `start-dicom-import-job` se inicia un trabajo de importación 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"
```

Salida:

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

Para obtener más información, consulte Iniciar un trabajo de importación en el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte S tartDICOMImport Job en AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte S tartDICOMImport Job en AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte S tartDICOMImport Job en AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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"]
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte S tartDICOMImport Job en AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`TagResource`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `TagResource`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Etiquetar un almacén de datos
- Etiquetar un conjunto de imágenes

CLI

AWS CLI

Ejemplo 1: etiquetado de un almacén de datos

En los siguientes ejemplos de código `tag-resource` se etiqueta un almacén de datos.

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

Este comando no genera ninguna salida.

Ejemplo 2: etiquetado de un conjunto de imágenes

En los siguientes ejemplos de código `tag-resource` se etiqueta un conjunto de imágenes.

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

Este comando no genera ninguna salida.

Para obtener más información, consulte Etiquetar recursos con AWS HealthImagingen el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte TagResourceen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte TagResourceen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte TagResourceen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en TagResourceAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`UntagResource`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `UntagResource`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Etiquetar un almacén de datos
- Etiquetar un conjunto de imágenes

CLI

AWS CLI

Ejemplo 1: eliminación de las etiquetas de un almacén de datos

En el siguiente ejemplo de código `untag-resource` se eliminan las etiquetas de un almacén de datos.

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

Este comando no genera ninguna salida.

Ejemplo 2: eliminación de las etiquetas de un conjunto de imágenes

En el siguiente ejemplo de código `untag-resource` se eliminan las etiquetas de un conjunto de imágenes.

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

Este comando no genera ninguna salida.

Para obtener más información, consulte Etiquetar recursos con AWS HealthImagingen el AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte UntagResourceen AWS CLI Referencia de comandos.

Java

SDKpara 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 API obtener más información, consulte UntagResourceen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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 API obtener más información, consulte UntagResourceen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para obtener API más información, consulte en UntagResourceAWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

`UpdateImageSetMetadata`Úselo con un AWS SDKo CLI

En los siguientes ejemplos de código, se muestra cómo utilizar `UpdateImageSetMetadata`.

CLI

AWS CLI

Ejemplo 1: Para insertar o actualizar un atributo en los metadatos del conjunto de imágenes

En el siguiente `update-image-set-metadata` ejemplo, se inserta o actualiza un atributo en los metadatos del conjunto de imágenes.

```
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
```

Contenido de `metadata-updates.json`

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

Salida:

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

Ejemplo 2: Para eliminar un atributo de los metadatos del conjunto de imágenes

En el siguiente `update-image-set-metadata` ejemplo, se elimina un atributo de los metadatos del conjunto de imágenes.

```
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
```

Contenido de `metadata-updates.json`

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

Salida:

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

Ejemplo 3: Para eliminar una instancia de los metadatos del conjunto de imágenes

En el siguiente `update-image-set-metadata` ejemplo, se elimina una instancia de los metadatos del conjunto de imágenes.

```
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
```

Contenido 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\": {}}}}}}"
    }
}
```

Salida:

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

```
}
```

Ejemplo 4: Para revertir un conjunto de imágenes a una versión anterior

El siguiente `update-image-set-metadata` ejemplo muestra cómo revertir un conjunto de imágenes a una versión anterior. CopyImageSet y UpdateImageSetMetadata las acciones crean nuevas versiones de conjuntos de imágenes.

```
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"}'
```

Salida:

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

Ejemplo 5: Para añadir un elemento de DICOM datos privados a una instancia

El siguiente `update-image-set-metadata` ejemplo muestra cómo añadir un elemento privado a una instancia específica dentro de un conjunto de imágenes. El DICOM estándar permite que los elementos de datos privados comuniquen información que no puede estar contenida en los elementos de datos estándar. Puede crear, actualizar y eliminar elementos de datos privados con esta UpdateImageSetMetadata acción.

```
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
```

Contenido 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\"}}}}}}"
    }
}
```

Salida:

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

Ejemplo 6: Para actualizar un elemento de DICOM datos privados en una instancia

El siguiente `update-image-set-metadata` ejemplo muestra cómo actualizar el valor de un elemento de datos privado que pertenece a una instancia dentro de un conjunto de imágenes.

```
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
```

Contenido 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\"}}}}}}"
    }
}
```

Salida:

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

Ejemplo 7: Para actualizar a SOPInstanceUID con el parámetro force

En el siguiente `update-image-set-metadata` ejemploSOPInstanceUID, se muestra cómo actualizar a mediante el parámetro force para anular las restricciones de los DICOM metadatos.

```
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
```

Contenido 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\"}}}}}}}"
    }
}
```

Salida:

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

Para obtener más información, consulte Actualización de los metadatos del conjunto de imágenes en AWS HealthImaging Guía para desarrolladores.

- Para API obtener más información, consulte UpdateImageSetMetadataen AWS CLI Referencia de comandos.

Java

SDKpara 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: insertar o actualizar un 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: eliminar un 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: eliminar una instancia.

```
                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: volver a una versión 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 API obtener más información, consulte UpdateImageSetMetadataen AWS SDK for Java 2.x APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara 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: inserta o actualiza un atributo y fuerza la actualización.

```
        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: eliminar un 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: eliminar una instancia.

```
        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: volver a una versión anterior.

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

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

- Para API obtener más información, consulte UpdateImageSetMetadataen AWS SDK for JavaScript APIReferencia.

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

Caso de uso #1: inserta o actualiza un 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: eliminar un 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: eliminar una instancia.

```
        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: volver a una versión anterior.

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

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

- Para API obtener más información, consulte UpdateImageSetMetadataen AWS SDKpara referencia de Python (Boto3). API

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

Escenarios de HealthImaging uso AWS SDKs

Los siguientes ejemplos de código muestran cómo implementar escenarios comunes en HealthImaging with AWS SDKs. Estos escenarios muestran cómo realizar tareas específicas mediante la invocación de múltiples funciones dentro de otras HealthImaging o combinadas con otras Servicios de AWS. Cada escenario incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código.

Los escenarios se centran en un nivel intermedio de experiencia para ayudarle a entender las acciones de servicio en su contexto.

Ejemplos

- Comience con los conjuntos HealthImaging de imágenes y los marcos de imágenes con un AWS SDK
- Etiquetar un banco HealthImaging de datos mediante un AWS SDK
- Etiquetar un conjunto HealthImaging de imágenes mediante un AWS SDK

Comience con los conjuntos HealthImaging de imágenes y los marcos de imágenes con un AWS SDK

Los siguientes ejemplos de código muestran cómo importar DICOM archivos y descargar marcos de imágenes en ellos HealthImaging.

La implementación está estructurada como una aplicación de línea de comandos de flujo de trabajo.

- Configure los recursos para una DICOM importación.
- Importe DICOM archivos a un almacén de datos.
- Recupere el conjunto de imágenes IDs para el trabajo de importación.
- Recupere el marco de imágenes IDs de los conjuntos de imágenes.
- Descargue, decodifique y verifique los marcos de imágenes.
- Eliminación de recursos.

C++

SDKpara C++

Crea un AWS CloudFormation pila con los recursos necesarios.

```
    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 DICOM los archivos al depósito de importación de 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 los DICOM archivos al almacén de datos de 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;
}
```

Obtenga los conjuntos de imágenes creados por el trabajo de DICOM importación.

```
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;
}
```

Obtenga información sobre los marcos de imágenes para los conjuntos de imágenes.

```
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();
}
```

Descargue, decodifique y verifique los marcos de imágenes.

```
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;
}
```

Eliminación de 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 API obtener más información, consulte los siguientes temas en AWS SDK for C++ APIReferencia.
  - DeleteImageSet
  - G etDICOMImport Job
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Un tartDICOMImport trabajo

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

index.js - Organice los pasos.

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
```

```
import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
```

```
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
```

```
}
```

deploy-steps.js - Despliegue los recursos.

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

```
export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const 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 },
);
```

dataset-steps.js - Copiar DICOM archivos.

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
```

```
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
```

```
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js - Comience a importar al almacén de datos.

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

import {
  ScenarioAction,
  ScenarioOutput,
```

```
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
```

```
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js - Obtenga el conjunto de imágenes. IDs

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
```

```
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);
```

```
export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js - Obtenga el marco de la imagenIDs.

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
```

```
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
```

```
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }
```

```
    return output;
  },
  { slow: false },
);
```

verify-steps.js - Verifica los marcos de las imágenes. Con la AWS HealthImaging Para la verificación se utilizó la biblioteca de verificación de datos de píxeles.

```
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}`,
                ),
              );
            }
          });
        });
      }
```

```
      }
    }
  },
);
```

clean-up-steps.js - Destruye los recursos.

```
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 API obtener más información, consulte los siguientes temas en AWS SDK for JavaScript APIReferencia.
  - DeleteImageSet
  - G etDICOMImport Job
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Un tartDICOMImport trabajo

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

Crea un AWS CloudFormation pila con los recursos necesarios.

```
    def deploy(self):
        """
        Deploys prerequisite resources used by the scenario. The resources are
        defined in the associated `setup.yaml` AWS CloudFormation script and are
deployed
        as a CloudFormation stack, so they can be easily managed and destroyed.
        """

        print("\t\tLet's deploy the stack for resource creation.")
        stack_name = q.ask("\t\tEnter a name for the stack: ", q.non_empty)

        data_store_name = q.ask(
            "\t\tEnter a name for the Health Imaging Data Store: ", q.non_empty
        )
```

```
        account_id = boto3.client("sts").get_caller_identity()["Account"]

        with open(
            "../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml"
        ) as setup_file:
            setup_template = setup_file.read()
        print(f"\t\tCreating {stack_name}.")
        stack = self.cf_resource.create_stack(
            StackName=stack_name,
            TemplateBody=setup_template,
            Capabilities=["CAPABILITY_NAMED_IAM"],
            Parameters=[
                {
                    "ParameterKey": "datastoreName",
                    "ParameterValue": data_store_name,
                },
                {
                    "ParameterKey": "userAccountID",
                    "ParameterValue": account_id,
                },
            ],
        )
        print("\t\tWaiting for stack to deploy. This typically takes a minute or
 two.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
        waiter.wait(StackName=stack.name)
        stack.load()
        print(f"\t\tStack status: {stack.stack_status}")

        outputs_dictionary = {
            output["OutputKey"]: output["OutputValue"] for output in
 stack.outputs
        }
        self.input_bucket_name = outputs_dictionary["BucketName"]
        self.output_bucket_name = outputs_dictionary["BucketName"]
        self.role_arn = outputs_dictionary["RoleArn"]
        self.data_store_id = outputs_dictionary["DatastoreID"]
        return stack
```

Copie DICOM los archivos al depósito de importación de 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 los DICOM archivos al almacén de datos de Amazon S3.

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def start_dicom_import_job(
        self,
        data_store_id,
        input_bucket_name,
        input_directory,
        output_bucket_name,
        output_directory,
        role_arn,
    ):
        """
        Routine which starts a HealthImaging import job.

        :param data_store_id: The HealthImaging data store ID.
        :param input_bucket_name: The name of the Amazon S3 bucket containing the
 DICOM files.
        :param input_directory: The directory in the S3 bucket containing the
 DICOM files.
        :param output_bucket_name: The name of the S3 bucket for the output.
```

```
        :param output_directory: The directory in the S3 bucket to store the
 output.
        :param role_arn: The ARN of the IAM role with permissions for the import.
        :return: The job ID of the import.
        """

        input_uri = f"s3://{input_bucket_name}/{input_directory}/"
        output_uri = f"s3://{output_bucket_name}/{output_directory}/"
        try:
            job = self.medical_imaging_client.start_dicom_import_job(
                jobName="examplejob",
                datastoreId=data_store_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_uri,
                outputS3Uri=output_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

Obtenga los conjuntos de imágenes creados por el trabajo de DICOM importación.

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client
```

```
    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
        """
        Retrieves the image sets created for an import job.

        :param datastore_id: The HealthImaging data store ID
        :param import_job_id: The import job ID
        :return: List of image set IDs
        """

        import_job = self.medical_imaging_client.get_dicom_import_job(
            datastoreId=datastore_id, jobId=import_job_id
        )

        output_uri = import_job["jobProperties"]["outputS3Uri"]

        bucket = output_uri.split("/")[2]
        key = "/".join(output_uri.split("/")[3:])

        # Try to get the manifest.
        retries = 3
        while retries > 0:
            try:
                obj = self.s3_client.get_object(
                    Bucket=bucket, Key=key + "job-output-manifest.json"
                )
                body = obj["Body"]
                break
            except ClientError as error:
                retries = retries - 1
                time.sleep(3)
        try:
            data = json.load(body)
            expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
            image_sets = expression.search(data)
        except json.decoder.JSONDecodeError as error:
            image_sets = import_job["jobProperties"]
```

```
        return image_sets


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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.medical_imaging_client.get_image_set(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set = self.medical_imaging_client.get_image_set(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
        except ClientError as err:
            logger.error(
                "Couldn't get image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return image_set
```

Obtenga información sobre los marcos de imágenes para los conjuntos de imágenes.

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""
```

```
    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_frames_for_image_set(self, datastore_id, image_set_id,
out_directory):
        """
        Get the image frames for an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param out_directory: The directory to save the file.
        :return: The image frames.
        """
        image_frames = []
        file_name = os.path.join(out_directory,
f"{image_set_id}_metadata.json.gzip")
        file_name = file_name.replace("/", "\\\\")
        self.get_image_set_metadata(file_name, datastore_id, image_set_id)
        try:
            with gzip.open(file_name, "rb") as f_in:
                doc = json.load(f_in)
            instances = jmespath.search("Study.Series.*.Instances[].*[]", doc)
            for instance in instances:
                rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
                rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)
                image_frames_json = jmespath.search("ImageFrames[][]", instance)
                for image_frame in image_frames_json:
                    checksum_json = jmespath.search(
                        "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",
                        image_frame,
```

```
                    )
                    image_frame_info = {
                        "imageSetId": image_set_id,
                        "imageFrameId": image_frame["ID"],
                        "rescaleIntercept": rescale_intercept,
                        "rescaleSlope": rescale_slope,
                        "minPixelValue": image_frame["MinPixelValue"],
                        "maxPixelValue": image_frame["MaxPixelValue"],
                        "fullResolutionChecksum": checksum_json["Checksum"],
                    }
                    image_frames.append(image_frame_info)
            return image_frames
        except TypeError:
            return {}
        except ClientError as err:
            logger.error(
                "Couldn't get image frames for image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        return image_frames


    def get_image_set_metadata(
        self, metadata_file, datastore_id, image_set_id, version_id=None
    ):
        """
        Get the metadata of an image set.

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """

        try:
            if version_id:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
```

```
            else:
                image_set_metadata =
 self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
            with open(metadata_file, "wb") as f:
                for chunk in
 image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)

        except ClientError as err:
            logger.error(
                "Couldn't get image metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Descargue, decodifique y verifique los marcos de imágenes.

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)
```

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

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


    def download_decode_and_check_image_frames(
        self, data_store_id, image_frames, out_directory
    ):
        """
        Downloads image frames, decodes them, and uses the checksum to validate
        the decoded images.

        :param data_store_id: The HealthImaging data store ID.
        :param image_frames: A list of dicts containing image frame information.
        :param out_directory: A directory for the downloaded images.
        :return: True if the function succeeded; otherwise, False.
        """
        total_result = True
        for image_frame in image_frames:
```

```
            image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
            self.get_pixel_data(
                image_file_path,
                data_store_id,
                image_frame["imageSetId"],
                image_frame["imageFrameId"],
            )

            image_array = self.jph_image_to_opj_bitmap(image_file_path)
            crc32_checksum = image_frame["fullResolutionChecksum"]
            # Verify checksum.
            crc32_calculated = zlib.crc32(image_array)
            image_result = crc32_checksum == crc32_calculated
            print(
                f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
 {image_result }"
            )
            total_result = total_result and image_result
        return total_result

    @staticmethod
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array
```

Eliminación de 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 Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
```

```
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.medical_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries


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

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

- Para API obtener más información, consulte los siguientes temas en AWS SDKpara referencia de Python (Boto3). API
  - DeleteImageSet
  - G etDICOMImport Job
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Un tartDICOMImport trabajo

> **Note**
>
> Hay más en marcha GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

Etiquetar un banco HealthImaging de datos mediante un AWS SDK

Los siguientes ejemplos de código muestran cómo etiquetar un banco HealthImaging de datos.

Java

SDKpara Java 2.x

Para etiquetar un almacén de datos

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

Función de utilidad para etiquetar un 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);
       }
   }
```

Para enumerar las etiquetas de almacenes de datos

```
                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());
                }
```

La función de utilidad para enumerar las etiquetas de un 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;
   }
```

Para desetiquetar un almacén de datos

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

La función de utilidad para eliminar la etiqueta de un 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 API obtener más información, consulte los siguientes temas en AWS SDK for Java 2.x APIReferencia.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

Para etiquetar un almacén de datos

```
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);
  }
```

Función de utilidad para etiquetar un 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;
};
```

Para enumerar las etiquetas de almacenes de datos

```
  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);
  }
```

La función de utilidad para enumerar las etiquetas de un 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;
};
```

Para desetiquetar un almacén de datos

```
  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);
  }
```

La función de utilidad para eliminar la etiqueta de un 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 API obtener más información, consulte los siguientes temas en AWS SDK for JavaScript APIReferencia.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

Para etiquetar un almacén de datos

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

Función de utilidad para etiquetar un 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
```

Para enumerar las etiquetas de almacenes de datos

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

La función de utilidad para enumerar las etiquetas de un 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"]
```

Para desetiquetar un almacén de datos

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

La función de utilidad para eliminar la etiqueta de un 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte los siguientes temas en AWS SDKpara referencia de Python (Boto3). API
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

Etiquetar un conjunto HealthImaging de imágenes mediante un AWS SDK

Los siguientes ejemplos de código muestran cómo etiquetar un conjunto HealthImaging de imágenes.

Java

SDKpara Java 2.x

Para etiquetar un conjunto de imágenes

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

Función de utilidad para etiquetar un 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);
       }
   }
```

Para enumerar las etiquetas de un conjunto de imágenes

```
                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());
                }
```

La función de utilidad para enumerar las etiquetas de un 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;
   }
```

Para desetiquetar un conjunto de imágenes

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                Collections.singletonList("Deployment"));
```

La función de utilidad para eliminar la etiqueta de un 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 API obtener más información, consulte los siguientes temas en AWS SDK for Java 2.x APIReferencia.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

JavaScript

SDKpara JavaScript (v3)

Para etiquetar un conjunto de imágenes

```
  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);
  }
```

Función de utilidad para etiquetar un 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;
};
```

Para enumerar las etiquetas de un conjunto de imágenes

```
  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);
  }
```

La función de utilidad para enumerar las etiquetas de un 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;
};
```

Para desetiquetar un conjunto de imágenes

```
  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);
  }
```

La función de utilidad para eliminar la etiqueta de un 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 API obtener más información, consulte los siguientes temas en AWS SDK for JavaScript APIReferencia.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Hay más información GitHub. Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Python

SDKpara Python (Boto3)

Para etiquetar un conjunto de imágenes

```
    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"})
```

Función de utilidad para etiquetar un 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
```

Para enumerar las etiquetas de un conjunto de imágenes

```
    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)
```

La función de utilidad para enumerar las etiquetas de un 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"]
```

Para desetiquetar un conjunto de imágenes

```
    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"])
```

La función de utilidad para eliminar la etiqueta de un 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
```

El siguiente código crea una instancia del objeto. MedicalImagingWrapper

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

- Para API obtener más información, consulte los siguientes temas en AWS SDKpara referencia de Python (Boto3). API
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Hay más información. GitHub Consulta el ejemplo completo y aprende a configurarlo y a ejecutarlo en AWS Repositorio de ejemplos de código.

Para obtener una lista completa de AWS SDKguías para desarrolladores y ejemplos de código, consulte HealthImaging Utilización con un AWS SDK. En este tema también se incluye información sobre cómo empezar y detalles sobre SDK las versiones anteriores.

Uso DICOMweb con AWS HealthImaging

Puede recuperar DICOM objetos AWS HealthImaging utilizando una representación de DICOMwebAPIs, que son servicios basados en la web que siguen el DICOM estándar de imágenes médicas. Esta funcionalidad le permite interoperar con sistemas que utilizan archivos binarios de la DICOM parte 10 y, al mismo tiempo, aprovechar las funciones nativas HealthImaging de la nube. Este capítulo se centra en cómo utilizar la implementación HealthImaging de DICOMweb servicios para devolver DICOMweb respuestas.

> **Importante**
>
> HealthImaging los trabajos de importación procesan binarios de DICOM instancias (`.dcm`archivos) y los transforman en conjuntos de imágenes. Utilice acciones nativas de HealthImaging la nube (APIs) para administrar conjuntos de imágenes. Utilice HealthImaging la representación de los DICOMweb servicios para devolver DICOMweb las respuestas. Los que APIs se enumeran en este capítulo están diseñados de conformidad con el DICOMwebestándar de imágenes médicas basadas en la web. Debido a que son representaciones de DICOMweb servicios, no se ofrecen a través de AWS CLI y AWS SDKs.

Tema

- Recuperación de DICOM datos de HealthImaging

Recuperación de DICOM datos de HealthImaging

AWS HealthImaging ofrece representaciones de `DICOMweb WADO-RS`servicios para recuperar una DICOM instancia, los metadatos de la DICOM instancia y los marcos de la DICOM instancia (datos de píxeles) de un almacén de HealthImaging datos. HealthImagingSus `DICOMweb WADO-RS` servicios ofrecen flexibilidad a la hora de recuperar los datos almacenados HealthImaging y proporcionan interoperabilidad con las aplicaciones antiguas.

> **Importante**
>
> HealthImaging los trabajos de importación procesan binarios de DICOM instancias (`.dcm`archivos) y los transforman en conjuntos de imágenes. Utilice acciones nativas de

HealthImaging la nube (APIs) para administrar conjuntos de imágenes. Utilice HealthImaging la representación de los DICOMweb servicios para devolver DICOMweb las respuestas. Los servicios que se enumeran en esta sección están diseñados de conformidad con el estándar DICOMweb(WADO-RS) de imágenes médicas basadas en la web. Como son representaciones de DICOMweb servicios, no se ofrecen a través AWS CLI de y. AWS SDKs

En la siguiente tabla se describen todas las HealthImaging representaciones de DICOMweb WADO los servicios -RS disponibles para recuperar datos. HealthImaging

HealthImaging representaciones de los servicios DICOMweb WADO -RS

Nombre	Descripción
`GetDICOMInstance`	Recupere una DICOM instancia (`.dcm`archivo) de un almacén de HealthImaging datos especificando la serie, el estudio y la instancia UIDs asociados a un recurso.
`GetDICOMInstanceMetadata`	Para recuperar los metadatos de la DICOM instancia (`.json`archivo) de una DICOM instancia en un banco de HealthImaging datos, especifique la serie, el estudio y la instancia UIDs asociados a un recurso.
`GetDICOMInstanceFrames`	Para recuperar fotogramas de imagen individua les o por lotes (`multipart` solicitud) de una DICOM instancia de un almacén de HealthIma ging datosUID, especifique los números de serieUID, estudioUIDs, instancia y fotograma asociados a un recurso.

Temas

- Obtener una DICOM instancia de HealthImaging
- Obtener metadatos de DICOM instancias de HealthImaging
- Obtener marcos de DICOM instancia de HealthImaging

Obtener una DICOM instancia de HealthImaging

Utilice la `GetDICOMInstance` acción para recuperar una DICOM instancia (`.dcm`archivo) de un HealthImaging banco de datos especificando la serie, el estudio y la instancia UIDs asociados al recurso. Puede especificar el conjunto de imágenes del que se debe recuperar un recurso de instancia proporcionando el ID del conjunto de imágenes como parámetro de consulta. Además, puedes elegir la sintaxis de transferencia para comprimir los DICOM datos, con soporte para uncompressed (ELE) o High-Throughput JPEG 2000 (). HTJ2K

> **Note**
>
> Con él`GetDICOMInstance`, puede interoperar con sistemas que utilizan binarios de la DICOM parte 10 y, al mismo tiempo, aprovechar las acciones nativas de la HealthImaging nube.

Para obtener una DICOM instancia () **`.dcm`**

1. Recopile valores HealthImaging `datastoreId` y `imageSetId` parametrice.
2. Utilice la `GetImageSetMetadata`acción con los valores de los `imageSetId` parámetros `datastoreId` y para recuperar los valores de metadatos asociados para `studyInstanceUIDseriesInstanceUID`, y`sopInstanceUID`. Para obtener más información, consulte Obtención de metadatos de conjuntos de imágenes.
3. Construya a URL para la solicitud utilizando los valores de `datastoreId` `studyInstanceUIDseriesInstanceUID`,`sopInstanceUID`, y`imageSetId`. Para ver la URL ruta completa en el siguiente ejemplo, desplázate sobre el botón Copiar. Tiene URL el siguiente formato:

   ```
   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. Prepara y envía tu solicitud. `GetDICOMInstance`utiliza una HTTP GET solicitud con el protocolo de AWS firma Signature versión 4. El siguiente ejemplo de código utiliza la herramienta de línea de `curl` comandos para obtener una DICOM instancia (`.dcm`archivo) 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**
>
> `transfer-syntax`UIDEs opcional y, si no se incluye, el valor predeterminado es Explicit VR Little Endian. Las sintaxis de transferencia compatibles incluyen:
>
> - Explicit VR Little Endian (ELE) (-) `1.2.840.10008.1.2.1` (predeterminado)
> - JPEG2000 de alto rendimiento con RPCL opciones de compresión de imágenes (solo sin pérdidas) - `1.2.840.10008.1.2.4.202`
>
> Para obtener más información, consulte Bibliotecas de decodificación HTJ2K para AWS HealthImaging.

Obtener metadatos de DICOM instancias de HealthImaging

Use la `GetDICOMInstanceMetadata` acción para recuperar los metadatos de una DICOM instancia en un HealthImaging banco de datos especificando la serie, el estudio y la instancia UIDs asociados al recurso. Para especificar el conjunto de imágenes del que se deben recuperar los metadatos de los recursos de la instancia, proporciona el ID del conjunto de imágenes como parámetro de consulta.

> **Note**
>
> Con élGetDICOMInstanceMetadata, puedes interoperar con sistemas que utilizan archivos binarios de la DICOM parte 10 y, al mismo tiempo, aprovechar las acciones nativas HealthImaging de la nube.

Para obtener los metadatos de la DICOM instancia () **.json**

1. Recopile valores HealthImaging datastoreId y imageSetId parametrice.
2. Utilice la GetImageSetMetadataacción con los valores de los imageSetId parámetros datastoreId y para recuperar los valores de metadatos asociados para studyInstanceUIDseriesInstanceUID, ysopInstanceUID. Para obtener más información, consulte Obtención de metadatos de conjuntos de imágenes.
3. Construya a URL para la solicitud utilizando los valores de datastoreId studyInstanceUIDseriesInstanceUID,sopInstanceUID, yimageSetId. Para ver la URL ruta completa en el siguiente ejemplo, desplázate sobre el botón Copiar. Tiene URL el siguiente formato:

   ```
   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. Prepara y envía tu solicitud. GetDICOMInstanceMetadatautiliza una HTTP GET solicitud con el protocolo de AWS firma Signature versión 4. El siguiente ejemplo de código utiliza la herramienta de línea de curl comandos para obtener los metadatos de la DICOM instancia (.jsonarchivo) 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'
```

Obtener marcos de DICOM instancia de HealthImaging

Utilice esta `GetDICOMInstanceFrames` acción para recuperar fotogramas de imagen individuales o por lotes (`multipart`solicitud) de una DICOM instancia de un HealthImaging data store especificando los números de serie UIDUID, estudioUIDs, instancia y fotograma asociados a un recurso. Puede especificar el conjunto de imágenes del que se deben recuperar los marcos de instancia proporcionando el ID del conjunto de imágenes como parámetro de consulta. Además, puedes elegir la sintaxis de transferencia para comprimir los datos del marco de la imagen, con compatibilidad con Uncompressed (ELE) o HighThroughput JPEG 2000 (). HTJ2K

> **Note**
>
> Con él`GetDICOMInstanceFrames`, puede interoperar con sistemas que utilizan archivos binarios de la DICOM parte 10 y, al mismo tiempo, aprovechar las acciones nativas de la HealthImaging nube.

Para obtener marcos de DICOM instancia () **`multipart`**

1. Recopile valores HealthImaging `datastoreId` y `imageSetId` parametrice.
2. Utilice la `GetImageSetMetadata`acción con los valores de los `imageSetId` parámetros `datastoreId` y para recuperar los valores de metadatos asociados para `studyInstanceUIDseriesInstanceUID`, y`sopInstanceUID`. Para obtener más información, consulte Obtención de metadatos de conjuntos de imágenes.
3. Determine los marcos de imagen que se van a recuperar de los metadatos asociados para formar el `frameList` parámetro. El `frameList` parámetro es una lista separada por comas de uno o más números de fotogramas no duplicados, en cualquier orden. Por ejemplo, el primer marco de imagen de los metadatos será el marco 1.
   - Solicitud de fotograma único: `/frames/1`
   - Solicitud de fotogramas múltiples: `/frames/1,2,3,4`
4. Construye a URL para la solicitud utilizando los valores de `datastoreIdstudyInstanceUID`,`seriesInstanceUID`, `sopInstanceUIDimageSetId`,

`yframeList`. Para ver la URL ruta completa en el siguiente ejemplo, desplázate sobre el botón Copiar. Tiene URL el siguiente formato:

```
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. Prepara y envía tu solicitud. `GetDICOMInstanceFrames`utiliza una HTTP GET solicitud con el protocolo de AWS firma Signature versión 4. El siguiente ejemplo de código utiliza la herramienta de línea de `curl` comandos para obtener marcos de imagen en una `multipart` respuesta desde 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**
>
> `transfer-syntax`UIDEs opcional y, si no se incluye, el valor predeterminado es Explicit VR Little Endian. Las sintaxis de transferencia compatibles incluyen:
>
> - Explicit VR Little Endian (ELE) (-) `1.2.840.10008.1.2.1` (predeterminado)
> - JPEG2000 de alto rendimiento con RPCL opciones de compresión de imágenes (solo sin pérdidas) - `1.2.840.10008.1.2.4.202`
>
>   Para obtener más información, consulte Bibliotecas de decodificación HTJ2K para AWS HealthImaging.

Supervisión de AWS HealthImaging

La supervisión y el registro son partes importantes del mantenimiento de la seguridad, la fiabilidad, la disponibilidad y el rendimiento de AWS HealthImaging. AWS proporciona las siguientes herramientas de registro y monitoreo para observar HealthImaging, informar cuando algo va mal y tomar medidas automáticas cuando sea apropiado:

- AWS CloudTrailcaptura las llamadas a la API y los eventos relacionados realizados por su AWS cuenta o en su nombre y entrega los archivos de registro a un bucket de Amazon S3 que especifique. Puede identificar qué usuarios y cuentas llamaron AWS, la dirección IP de origen desde la que se realizaron las llamadas y cuándo se produjeron. Para obtener más información, consulte la Guía del usuario de AWS CloudTrail.
- Amazon CloudWatch monitorea tus AWS recursos y las aplicaciones en las que AWS ejecutas en tiempo real. Puede recopilar métricas y realizar un seguimiento de las métricas, crear paneles personalizados y definir alarmas que le advierten o que toman medidas cuando una métrica determinada alcanza el umbral que se especifique. Por ejemplo, puede CloudWatch hacer un seguimiento del uso de la CPU u otras métricas de sus instancias de Amazon EC2 y lanzar automáticamente nuevas instancias cuando sea necesario. Para obtener más información, consulta la Guía del CloudWatch usuario de Amazon.
- Amazon EventBridge es un servicio de bus de eventos sin servidor que facilita la conexión de sus aplicaciones con datos de diversas fuentes. EventBridge ofrece un flujo de datos en tiempo real desde sus propias aplicaciones, aplicaciones de oftware-as-a S-Service (SaaS) AWS y servicios, y dirige esos datos a destinos como Lambda. Esto le permite monitorear los eventos que ocurren en los servicios y crear arquitecturas basadas en eventos. Para obtener más información, consulta la Guía del EventBridge usuario de Amazon.

Temas

Uso AWS CloudTrail con HealthImaging

AWS HealthImaging está integrado con AWS CloudTrail un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un AWS servicio en HealthImaging. CloudTrail captura todas las llamadas a la API HealthImaging como eventos. Las llamadas capturadas incluyen llamadas desde la HealthImaging consola y llamadas en código a las operaciones de la HealthImaging API. Si crea una ruta, puede activar la entrega continua de CloudTrail eventos a un bucket de Amazon S3, incluidos los eventos de HealthImaging. Si no configura una ruta, podrá ver los eventos más recientes en la CloudTrail consola, en el historial de eventos. Con la información recopilada por usted CloudTrail, puede determinar el destinatario de la solicitud HealthImaging, la dirección IP desde la que se realizó la solicitud, quién la realizó, cuándo se realizó y detalles adicionales.

Para obtener más información CloudTrail, consulte la Guía AWS CloudTrail del usuario.

Creación de un registro de seguimiento

CloudTrail se activa Cuenta de AWS cuando creas la cuenta. Cuando se produce una actividad en HealthImaging, esa actividad se registra en un CloudTrail evento junto con otros eventos de AWS servicio en el historial de eventos. Puede ver, buscar y descargar eventos recientes en su Cuenta de AWS. Para obtener más información, consulte Visualización de eventos con el historial de CloudTrail eventos.

> **Note**
>
> Para ver el historial de CloudTrail eventos de AWS HealthImaging en AWS Management Console, vaya al menú Lookup attributes, seleccione Event Source y elija`medical-imaging.amazonaws.com`.

Para obtener un registro continuo de sus eventos Cuenta de AWS, incluidos los eventos de su HealthImaging organización, cree una ruta. Un rastro permite CloudTrail entregar archivos de registro a un bucket de Amazon S3. De forma predeterminada, cuando se crea un registro de seguimiento en la consola, el registro de seguimiento se aplica a todas las Regiones de AWS. La ruta registra los eventos de todas las regiones de la AWS partición y envía los archivos de registro al bucket de Amazon S3 que especifique. Además, puede configurar otros AWS servicios para analizar más a fondo los datos de eventos recopilados en los CloudTrail registros y actuar en función de ellos. Para más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- CloudTrail servicios e integraciones compatibles
- Configuración de las notificaciones de Amazon SNS para CloudTrail
- Recibir archivos de CloudTrail registro de varias regiones y recibir archivos de CloudTrail registro de varias cuentas

> **Note**
>
> AWS HealthImaging admite dos tipos de CloudTrail eventos: eventos de administración y eventos de datos. Los eventos de administración son los eventos generales que genera cada AWS servicio, incluidos los siguientes HealthImaging: De forma predeterminada, el registro se aplica a los eventos de administración de cada llamada a la HealthImaging API que lo tenga habilitado. Los eventos de datos se facturan y, por lo general, se reservan para las API que tienen un alto número de transacciones por segundo (tps), por lo que puede optar por no tener CloudTrail registros por motivos de costes.
> Con HealthImaging, todas las acciones de API enumeradas en la Referencia de HealthImaging API de AWS se consideran eventos de administración, con la excepción de `GetImageFrame`. La `GetImageFrame` acción se incorpora CloudTrail como un evento de datos y, por lo tanto, debe estar habilitada. Para obtener más información, consulte Registro de eventos de datos en la Guía del usuario de AWS CloudTrail .

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario lo ayuda a determinar lo siguiente:

- Si la solicitud se realizó con credenciales de usuario root o AWS Identity and Access Management (IAM).
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro AWS servicio.

Para obtener más información, consulte el CloudTrail `userIdentity`elemento.

Descripción de las entradas de los entradas de registro

Un rastro es una configuración que permite la entrega de eventos como archivos de registro a un bucket de Amazon S3 que usted especifique. CloudTrail Los archivos de registro contienen una o más entradas de registro. Un evento representa una solicitud única de cualquier fuente e incluye información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. CloudTrail Los archivos de registro no son un registro ordenado de las llamadas a la API pública, por lo que no aparecen en ningún orden específico.

En el siguiente ejemplo, se muestra una entrada de CloudTrail registro HealthImaging que demuestra la `GetDICOMImportJob` acción.


```
{
    "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"
}
```


Uso de Amazon CloudWatch con HealthImaging

Puede monitorizar AWS HealthImaging mediante CloudWatch, que recopila datos sin procesar y los procesa para convertirlos en métricas legibles prácticamente en tiempo real. Estas estadísticas se mantienen durante 15 meses, de modo que pueda consultar información histórica y disponer de una mejor perspectiva sobre el desempeño de su aplicación web o servicio. También puede establecer alarmas que vigilen determinados umbrales y enviar notificaciones o realizar acciones cuando se cumplan dichos umbrales. Para obtener más información, consulta la Guía del CloudWatch usuario de Amazon.

> **Note**
>
> Se muestran las métricas de todas HealthImaging las API.

En las siguientes tablas se muestran las métricas y las dimensiones de HealthImaging. Cada una se presenta como un recuento de frecuencias para un rango de datos especificado por el usuario.

Métricas

Métricas	Descripción
Conteo de llamadas	El número de llamadas a las API. que se puede informar, o bien para la cuenta, o bien para un almacén de datos específico. Unidades: recuento Estadísticas válidas: Sum, Count Dimensiones: funcionamiento, ID del almacén de datos, tipo de almacén de datos

Puede obtener métricas HealthImaging con la AWS Management Console AWS CLI, la o la CloudWatch API. Puede usar la CloudWatch API a través de uno de los kits de desarrollo de software (SDK) de Amazon AWS o las herramientas de CloudWatch API. La HealthImaging consola muestra gráficos basados en los datos sin procesar de la CloudWatch API.

Debe tener los CloudWatch permisos adecuados para poder realizar la supervisión HealthImaging CloudWatch. Para obtener más información, consulte la sección Gestión de identidades y accesos CloudWatch en la Guía del CloudWatch usuario.

Visualización de HealthImaging las métricas

Para ver las métricas (CloudWatch consola)

1. Inicie sesión en la CloudWatch consola AWS Management Console y ábrala.
2. En Métricas, elija Todas las métricas e Imágenes de AWS/Medical.
3. Elija la dimensión, un nombre de métrica y, a continuación, Add to graph (Añadir al gráfico).
4. Elija un valor para el intervalo de fechas. El recuento de las métricas del intervalo de fechas seleccionado se muestra en el gráfico.

Crear una alarma mediante CloudWatch

Una CloudWatch alarma vigila una única métrica durante un período de tiempo específico y realiza una o más acciones: enviar una notificación a un tema del Amazon Simple Notification Service

(Amazon SNS) o a una política de Auto Scaling. La acción o las acciones se basan en el valor de la métrica en relación con un umbral determinado durante un número de períodos de tiempo que usted especifique. CloudWatch también puede enviarle un mensaje de Amazon SNS cuando la alarma cambie de estado.

CloudWatch las alarmas invocan acciones solo cuando el estado cambia y ha persistido durante el período que usted especifique. Para obtener más información, consulte Uso CloudWatch de alarmas.

Uso de Amazon EventBridge con HealthImaging

Amazon EventBridge es un servicio sin servidor que utiliza eventos para conectar los componentes de la aplicación entre sí, lo que facilita la creación de aplicaciones escalables basadas en eventos. La base EventBridge es crear reglas que dirijan los eventos a los objetivos. AWS HealthImaging proporciona una entrega duradera de los cambios de estado a EventBridge. Para obtener más información, consulta ¿Qué es Amazon EventBridge? en la Guía del EventBridge usuario de Amazon.

Temas

- HealthImaging eventos enviados a EventBridge
- HealthImaging estructura y ejemplos de eventos

HealthImaging eventos enviados a EventBridge

En la siguiente tabla se enumeran todos los HealthImaging eventos enviados a EventBridge para su procesamiento.

HealthImaging tipo de evento	Estado
eventos del almacén de datos	
Creación de un almacén de datos	`CREATING`
Falló la creación del almacén de datos	`CREATE_FAILED`
Se creó el almacén de datos	`ACTIVE`
Eliminación del almacén de datos	`DELETING`

HealthImaging tipo de evento	Estado
Almacén de datos eliminado	`DELETED`
Para obtener más información, consulte DataStoreStatus en la AWS API Reference. HealthImaging	
Importe eventos de trabajo	
Trabajo de importación enviado	`SUBMITTED`
Importar trabajo en curso	`IN_PROGRESS`
Trabajo de importación completado	`COMPLETED`
Error al importar el trabajo	`FAILED`
Para obtener más información, consulte JobStatus en la AWS HealthImaging API Reference.	
Eventos del conjunto de imágenes	
Conjunto de imágenes creado	`CREATED`
Copia del conjunto de imágenes	`COPYING`
Copia de conjuntos de imágenes con acceso de solo lectura	`COPYING_WITH_READ_ONLY_ACCESS`
Conjunto de imágenes copiado	`COPIED`
Falló la copia del conjunto de imágenes	`COPY_FAILED`
Actualización del conjunto de imágenes	`UPDATING`
Conjunto de imágenes actualizado	`UPDATED`
Falló la actualización del conjunto de imágenes	`UPDATE_FAILED`
Eliminación del conjunto de imágenes	`DELETING`

HealthImaging tipo de evento	Estado
Conjunto de imágenes eliminado	`DELETED`

Para obtener más información, consulte ImageSetWorkflowStatusla referencia de la HealthImaging API de AWS.

HealthImaging estructura y ejemplos de eventos

HealthImaging los eventos son objetos con estructura JSON que también contienen detalles de metadatos. Puede utilizar los metadatos como entrada para recrear un evento o para obtener más información. Todos los campos de metadatos asociados se muestran en una tabla bajo los ejemplos de código de los siguientes menús. Para obtener más información, consulta la referencia a la estructura de eventos en la Guía del EventBridge usuario de Amazon.

> **Note**
>
> El `source` atributo de las estructuras de HealthImaging eventos es`aws.medical-imaging`.

Eventos del almacén de datos

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 del almacén de datos: descripciones de metadatos

Nombre	Tipo	Descripción
`version`	cadena	La versión EventBridge del esquema de eventos.
`id`	cadena	El UUID de la versión 4 generado para cada evento.
`detail-type`	cadena	El tipo de evento que se envía.
`source`	cadena	Identifica el servicio que generó el evento.
`account`	cadena	El ID de cuenta de AWS de 12 dígitos del propietario del almacén de datos.
`time`	cadena	La hora a la que ocurrió el evento.
`region`	cadena	Identifica la AWS región del banco de datos.
`resources`	matriz (cadena)	Una matriz JSON que contiene el ARN del banco de datos.
`detail`	objeto	Un objeto JSON que contiene información sobre el evento.

Nombre	Tipo	Descripción
`detail.imagingVersion`	cadena	El ID de versión que rastrea los cambios en el esquema HealthImaging de detalles del evento.
`detail.datastoreId`	cadena	El ID del almacén de datos asociado al evento de cambio de estado.
`detail.datastoreName`	cadena	El nombre del almacén de datos.
`detail.datastoreStatus`	cadena	El estado actual del almacén de datos.

Importar eventos de trabajo

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 trabajo: descripciones de metadatos

Nombre	Tipo	Descripción
`version`	cadena	La versión EventBridge del esquema de eventos.

Nombre	Tipo	Descripción
`id`	cadena	El UUID de la versión 4 generado para cada evento.
`detail-type`	cadena	El tipo de evento que se envía.
`source`	cadena	Identifica el servicio que generó el evento.
`account`	cadena	El ID de cuenta de AWS de 12 dígitos del propietario del almacén de datos.
`time`	cadena	La hora a la que ocurrió el evento.
`region`	cadena	Identifica la AWS región del banco de datos.
`resources`	matriz (cadena)	Una matriz JSON que contiene el ARN del banco de datos.
`detail`	objeto	Un objeto JSON que contiene información sobre el evento.
`detail.imagingVersion`	cadena	El ID de versión que rastrea los cambios en el esquema HealthImaging de detalles del evento.
`detail.datastoreId`	cadena	El banco de datos que generó el evento de cambio de estado.

Nombre	Tipo	Descripción
`detail.jobId`	cadena	El identificador del trabajo de importación asociado al evento de cambio de estado.
`detail.jobName`	cadena	El nombre del trabajo de importación.
`detail.jobStatus`	cadena	El estado actual del trabajo.
`detail.inputS3Uri`	cadena	La ruta del prefijo de entrada para el depósito S3 que contiene los archivos DICOM que se van a importar.
`detail.outputS3Uri`	cadena	El prefijo de salida del depósito de S3 en el que se cargarán los resultados del trabajo de importación de DICOM.

Eventos del conjunto de imágenes

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",
```

```
      "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",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING"
   }
}
```

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",
      "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",
      "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",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "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",
      "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",
      "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",
      "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",
      "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",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "DELETED",
      "imageSetWorkflowStatus": "DELETED"
   }
}
```

Eventos del conjunto de imágenes: descripciones de metadatos

Nombre	Tipo	Descripción
`version`	cadena	La versión EventBridge del esquema de eventos.
`id`	cadena	El UUID de la versión 4 generado para cada evento.
`detail-type`	cadena	El tipo de evento que se envía.
`source`	cadena	Identifica el servicio que generó el evento.
`account`	cadena	El ID de cuenta de AWS de 12 dígitos del propietario del almacén de datos.
`time`	cadena	La hora a la que ocurrió el evento.
`region`	cadena	Identifica la AWS región del banco de datos.
`resources`	matriz (cadena)	Una matriz JSON que contiene el ARN del conjunto de imágenes.
`detail`	objeto	Un objeto JSON que contiene información sobre el evento.
`detail.imagingVersion`	cadena	El ID de versión que rastrea los cambios en el esquema HealthImaging de detalles del evento.

Nombre	Tipo	Descripción
`detail.datastoreId`	cadena	El ID del almacén de datos que generó el evento de cambio de estado.
`detail.imagesetId`	cadena	El ID del conjunto de imágenes asociado al evento de cambio de estado.
`detail.imageSetState`	cadena	El estado actual del conjunto de imágenes.
`detail.imageSetWorkflowStatus`	cadena	El estado actual del flujo de trabajo del conjunto de imágenes.

Seguridad en AWS HealthImaging

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de los centros de datos y las arquitecturas de red diseñados para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre AWS usted y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la Nube de AWS. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWS programas de de . Para obtener más información sobre los programas de cumplimiento aplicables AWS HealthImaging, consulte AWS Servicios incluidos en el ámbito de aplicación por programa de conformidad y AWS servicios incluidos .
- Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice. Usted también es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza HealthImaging. Los siguientes temas muestran cómo configurarlo HealthImaging para cumplir sus objetivos de seguridad y conformidad. También aprenderá a utilizar otros AWS servicios que le ayudan a supervisar y proteger sus HealthImaging recursos.

Temas

Protección de datos en AWS HealthImaging

La AWS modelo de responsabilidad compartida modelo de se aplica a la protección de datos en AWS HealthImaging. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global en la que se ejecutan todos los Nube de AWS. Usted es responsable de mantener el control sobre el contenido que está alojado en esta infraestructura. También es responsable de las tareas de configuración y administración de la seguridad del Servicios de AWS que utilices. Para obtener más información sobre la privacidad de los datos, consulte la sección Privacidad de datos FAQ. Para obtener información sobre la protección de datos en Europa, consulte la AWS Modelo de responsabilidad compartida y entrada de GDPR blog sobre AWS Blog de seguridad.

Para fines de protección de datos, le recomendamos que proteja Cuenta de AWS credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utilice la autenticación multifactorial (MFA) con cada cuenta.
- UtiliceSSL/TLSpara comunicarse con AWS recursos. Necesitamos TLS 1.2 y recomendamos TLS 1.3.
- Configure API y registre la actividad de los usuarios con AWS CloudTrail. Para obtener información sobre el uso de CloudTrail senderos para capturar AWS actividades, consulte Trabajar con CloudTrail senderos en AWS CloudTrail Guía del usuario.
- Uso AWS soluciones de cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita entre FIPS 140 y 3 módulos criptográficos validados para acceder AWS a través de una interfaz de línea de comandos oAPI, utilice un FIPS punto final. Para obtener más información sobre los FIPS puntos finales disponibles, consulte la Norma Federal de Procesamiento de Información (FIPS) 140-3.

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como, por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabaja con u otros HealthImaging Servicios de

AWS utilizando la consolaAPI, AWS CLI, o AWS SDKs. Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, le recomendamos encarecidamente que no incluya información sobre las credenciales URL para validar su solicitud a ese servidor.

Temas

Cifrado de datos

Con AWS HealthImaging él, puede añadir una capa de seguridad a sus datos almacenados en la nube, proporcionando funciones de cifrado escalables y eficientes. Entre ellos se incluyen:

- Las capacidades de cifrado de datos en reposo están disponibles en la mayoría AWS servicios
- Opciones flexibles de administración de claves, que incluyen AWS Key Management Service, con las que puede elegir si desea tener AWS administre las claves de cifrado o mantenga un control total sobre sus propias claves.
- AWS poseído AWS KMS claves de cifrado
- Colas de mensajes cifrados para la transmisión de datos confidenciales mediante cifrado del lado del servidor () SSE para Amazon SQS

Además, AWS le permite APIs integrar el cifrado y la protección de datos con cualquiera de los servicios que desarrolle o implemente en un AWS entorno.

Cifrado en reposo

HealthImaging proporciona cifrado de forma predeterminada para proteger los datos confidenciales de los clientes en reposo mediante el uso de un servicio propiedad AWS KMS clave.

Cifrado en tránsito

HealthImaging usa TLS 1.2 para cifrar los datos en tránsito a través del terminal público y a través de los servicios de backend.

Administración de claves

AWS KMS las claves (KMSclaves) son el recurso principal en AWS Key Management Service. También puede generar claves de datos para usarlas fuera de AWS KMS.

AWS KMSclave propia

HealthImaging utiliza estas claves de forma predeterminada para cifrar automáticamente la información potencialmente confidencial, como los datos de identificación personal o los datos de Private Health Information (PHI) en reposo. AWS KMSlas claves propias no se almacenan en tu cuenta. Forman parte de una colección de KMS llaves que AWS posee y administra para su uso en múltiples AWS cuentas. AWS los servicios pueden usar AWS KMSclaves propias para proteger sus datos. No puede ver, administrar ni usar AWS poseía KMS claves o auditaba su uso. Sin embargo, no es necesario que realice ninguna acción ni que cambie programas para proteger las claves que cifran sus datos.

No se te cobra una cuota mensual ni una cuota de uso si utilizas AWS KMSllaves propias, y no cuentan en contra AWS KMS cuotas para tu cuenta. Para obtener más información, consulte las claves AWS propias en la AWS Key Management Service Guía para desarrolladores.

Claves KMS administradas por el cliente

HealthImaging admite el uso de una KMS clave simétrica administrada por el cliente que usted crea, posee y administra para agregar una segunda capa de cifrado sobre la existente AWS cifrado propio. Como usted tiene el control total de esta capa de cifrado, puede realizar tareas como las siguientes:

- Establecer y mantener políticas, IAM políticas y subvenciones clave
- Rotar el material criptográfico
- Habilitar y deshabilitar políticas de claves
- Agregar etiquetas.
- Crear alias de clave
- Programar la eliminación de claves

También se puede utilizar CloudTrail para realizar un seguimiento de las solicitudes que se HealthImaging envían a AWS KMS en tu nombre. Adicional AWS KMS se aplican cargos. Para obtener más información, consulte Claves administradas por el cliente en AWS Key Management Service Guía para desarrolladores.

Creación de una clave administrada por el cliente

Puede crear una clave simétrica gestionada por el cliente mediante el AWS Management Console o el AWS KMS APIs. Para obtener más información, consulte Creación de KMS claves de cifrado simétricas en AWS Key Management Service Guía para desarrolladores.

Las políticas de clave controlan el acceso a la clave administrada por el cliente. Cada clave administrada por el cliente debe tener exactamente una política de clave, que contiene instrucciones que determinan quién puede usar la clave y cómo puede utilizarla. Cuando crea la clave administrada por el cliente, puede especificar una política de clave. Para obtener más información, consulte Administrar el acceso a las claves administradas por el cliente en la AWS Key Management Service Guía para desarrolladores.

Para utilizar la clave gestionada por el cliente con sus HealthImaging recursos, la política de claves debe permitir CreateGrant las operaciones de kms:. Esto añade una concesión a una clave gestionada por el cliente que controla el acceso a una KMS clave específica, lo que permite al usuario acceder a las operaciones de subvención HealthImaging requeridas. Para obtener más información, consulte Subvenciones en AWS KMS en la AWS Key Management Service Guía para desarrolladores.

Para utilizar la KMS clave gestionada por el cliente con HealthImaging los recursos, la política clave debe permitir las siguientes API operaciones:

- `kms:DescribeKey` proporciona los detalles necesarios de la clave administrada por el cliente para validar la clave. Esto es necesario para todas las operaciones.
- `kms:GenerateDataKey` proporciona acceso a los recursos de cifrado en reposo para todas las operaciones de escritura.
- `kms:Decrypt` proporciona acceso a las operaciones de lectura o búsqueda de recursos cifrados.
- `kms:ReEncrypt*` proporciona acceso para volver a cifrar los recursos.

El siguiente es un ejemplo de declaración de política que permite a un usuario crear un almacén de datos cifrado con esa clave e interactuar con él: HealthImaging

```
{
    "Sid": "Allow access to create data stores and perform CRUD and search in
 HealthImaging",
    "Effect": "Allow",
    "Principal": {
```

```
        "Service": [
            "medical-imaging.amazonaws.com"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:EncryptionContext:kms-arn": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f",
            "kms:EncryptionContext:aws:medical-imaging:datastoreId": "datastoreId"
        }
    }
}
```

IAMPermisos necesarios para usar una KMS clave administrada por el cliente

Al crear un almacén de datos con AWS KMS el cifrado se activa mediante una KMS clave gestionada por el cliente, por lo que se requieren permisos tanto para la política de claves como para la IAM política del usuario o rol que crea el almacén de HealthImaging datos.

Para obtener más información sobre las políticas clave, consulte Habilitar IAM las políticas en la AWS Key Management Service Guía para desarrolladores.

El IAM usuario, IAM el rol o AWS la cuenta que crea tus repositorios debe tener permisos para `kms:CreateGrant`kms:GenerateDataKey,`kms:RetireGrant`, y `kms:Decrypt`kms:ReEncrypt*, además de los permisos necesarios para AWS HealthImaging.

¿Cómo se HealthImaging utilizan las subvenciones en AWS KMS

HealthImaging requiere una concesión para utilizar la KMS clave gestionada por el cliente. Al crear un almacén de datos cifrado con una KMS clave gestionada por el cliente, HealthImaging crea una concesión en tu nombre enviando una CreateGrantsolicitud a AWS KMS. Becas en AWS KMS se utilizan para dar HealthImaging acceso a una KMS clave de la cuenta de un cliente.

Las subvenciones que se HealthImaging crean en su nombre no deben revocarse ni retirarse. Si revoca o retira la subvención que le da HealthImaging permiso para usar la AWS KMS las claves de su cuenta, HealthImaging no pueden acceder a estos datos, cifrar los nuevos recursos de imágenes introducidos en el almacén de datos o descifrarlos cuando se extraen. Al revocar o retirar

una subvención HealthImaging, el cambio se produce inmediatamente. Para revocar derechos de acceso, debe eliminar el almacén en lugar de revocar la concesión. Cuando se elimina un almacén de datos, HealthImaging retira las subvenciones en tu nombre.

Supervisión de las claves de cifrado para HealthImaging

Se puede utilizar CloudTrail para realizar un seguimiento de las solicitudes que se HealthImaging envían a AWS KMS en tu nombre cuando utilices una KMS clave gestionada por el cliente. Las entradas del CloudTrail registro se muestran `medical-imaging.amazonaws.com` en el `userAgent` campo para distinguir claramente las solicitudes realizadas por HealthImaging.

Los siguientes ejemplos son CloudTrail eventos para `CreateGrant`,`GenerateDataKey`,`Decrypt`, y `DescribeKey` para monitorear AWS KMS operaciones solicitadas HealthImaging para acceder a los datos cifrados por la clave gestionada por el cliente.

A continuación, se muestra cómo se utiliza `CreateGrant` para permitir el acceso HealthImaging a una KMS clave proporcionada por el cliente, lo que HealthImaging permite utilizar esa KMS clave para cifrar todos los datos inactivos del cliente.

Los usuarios no están obligados a crear sus propias subvenciones. HealthImaging crea una subvención en tu nombre enviando una `CreateGrant` solicitud a AWS KMS. Becas en AWS KMS se utilizan para dar HealthImaging acceso a un AWS KMS clave en una cuenta de cliente.


```
{
    "Grants": [
        {
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "DescribeKey"
            ],
            "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
            "Name": "0a74e6ad2aa84b74a22fcd3efac1eaa8",
            "RetiringPrincipal": "AWS Internal",
            "GranteePrincipal": "AWS Internal",
            "GrantId":
 "0da169eb18ffd3da8c0eebc9e74b3839573eb87e1e0dce893bb544a34e8fbaaf",
            "IssuingAccount": "AWS Internal",
            "CreationDate": 1685050229.0,
```

```
            "Constraints": {
                "EncryptionContextSubset": {
                    "kms-arn": "arn:aws:kms:us-
west-2:824333766656:key/2fe3c119-792d-4b99-822f-b5841e1181d1"
                }
            }
        },
        {
            "Operations": [
                "GenerateDataKey",
                "CreateGrant",
                "RetireGrant",
                "DescribeKey"
            ],
            "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
            "Name": "2023-05-25T21:30:17",
            "RetiringPrincipal": "AWS Internal",
            "GranteePrincipal": "AWS Internal",
            "GrantId":
 "8229757abbb2019555ba64d200278cedac08e5a7147426536fcd1f4270040a31",
            "IssuingAccount": "AWS Internal",
            "CreationDate": 1685050217.0,
        }
    ]
}
```

Los ejemplos siguientes muestran cómo utilizar `GenerateDataKey` para garantizar que los usuarios tengan los permisos necesarios para cifrar los datos antes de almacenarlos.

```
{
    "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"
}
```


El siguiente ejemplo muestra cómo se HealthImaging llama a la `Decrypt` operación para utilizar la clave de datos cifrados almacenada para acceder a los datos cifrados.


```
{
    "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"
}
```

En el siguiente ejemplo se muestra cómo se HealthImaging utiliza la `DescribeKey` operación para comprobar si AWS KMS propiedad del cliente AWS KMS la clave está en un estado utilizable y sirve para ayudar al usuario a solucionar problemas si no funciona.

```
{
    "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"
}
```

Más información

Los siguientes recursos proporcionan más información sobre el cifrado de datos en reposo y se encuentran en el AWS Key Management Service Guía para desarrolladores.

- AWS KMS conceptos
- Mejores prácticas de seguridad para AWS KMS

Privacidad del tráfico de red

El tráfico está protegido tanto entre HealthImaging las aplicaciones locales como entre Amazon S3 HealthImaging y Amazon S3. Tráfico entre y HealthImaging AWS Key Management Service usa HTTPS de forma predeterminada.

- AWS HealthImaging es un servicio regional disponible en las regiones EE.UU. Este (Norte de Virginia), EE.UU. Oeste (Oregón), Europa (Irlanda) y Asia Pacífico (Sídney).
- Para el tráfico entre HealthImaging los buckets de Amazon S3, Transport Layer Security (TLS) cifra los objetos en tránsito entre Amazon HealthImaging S3 HealthImaging y entre las aplicaciones de los clientes que acceden a él, solo debe permitir las conexiones cifradas a través de HTTPS (TLS) utilizando las políticas de bucket de Amazon `aws:SecureTransport condition`S3. IAM Aunque HealthImaging actualmente utiliza el punto de conexión público para acceder a los datos de los buckets de Amazon S3, esto no significa que los datos atraviesen la Internet pública. Todo el tráfico entre Amazon S3 HealthImaging y Amazon S3 se enruta a través del AWS red y se cifra medianteTLS.

Identity and Access Management para AWS HealthImaging

AWS Identity and Access Management (IAM) es un Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a AWS recursos. IAMlos administradores controlan quién puede autenticarse (iniciar sesión) y quién está autorizado (tiene permisos) para usar HealthImaging los recursos. IAMes un Servicio de AWS que puede utilizar sin coste adicional.

Temas

Público

¿Cómo se usa AWS Identity and Access Management (IAM) difiere según el trabajo en el que se realice HealthImaging.

Usuario del servicio: si utiliza el HealthImaging servicio para realizar su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más HealthImaging funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una función en HealthImaging, consulteSolución de problemas de identidad y acceso en AWS HealthImaging .

Administrador de servicios: si está a cargo de HealthImaging los recursos de su empresa, probablemente tenga acceso total a ellos HealthImaging. Su trabajo consiste en determinar a qué HealthImaging funciones y recursos deben acceder los usuarios del servicio. A continuación, debe enviar solicitudes a su IAM administrador para cambiar los permisos de los usuarios del servicio. Revise la información de esta página para comprender los conceptos básicos deIAM. Para obtener más información sobre cómo su empresa puede utilizar IAM con HealthImaging, consulte¿Cómo AWS HealthImaging funciona con IAM.

IAMadministrador: si es IAM administrador, puede que desee obtener más información sobre cómo puede redactar políticas para administrar el acceso a ellas HealthImaging. Para ver ejemplos de políticas HealthImaging basadas en la identidad que puede utilizarIAM, consulte. Ejemplos de políticas basadas en la identidad para AWS HealthImaging

Autenticación con identidades

La autenticación es la forma de iniciar sesión en AWS utilizando tus credenciales de identidad. Debe estar autenticado (haber iniciado sesión en AWS) como Usuario raíz de la cuenta de AWS, como IAM usuario o asumiendo un IAM rol.

Puede iniciar sesión en AWS como identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios de (IAMIdentity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, el administrador configuró previamente la federación de identidades mediante roles. IAM Cuando accedes AWS al usar la federación, está asumiendo un rol de manera indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en AWS Management Console o el AWS portal de acceso. Para obtener más información sobre cómo iniciar sesión en AWS, consulta Cómo iniciar sesión en tu Cuenta de AWS en la AWS Sign-In Guía del usuario.

Si accedes AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no usa AWS herramientas, debe firmar las solicitudes usted mismo. Para obtener más información sobre cómo usar el método recomendado para firmar las solicitudes usted mismo, consulte Firmar AWS APIsolicitudes en la Guía IAM del usuario.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo: AWS recomienda que utilice la autenticación multifactorial (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactorial en la AWS IAM Identity Center Guía del usuario y Uso de la autenticación multifactorial () MFA en AWS en la Guía del usuario de IAM.

Cuenta de AWS usuario raíz

Al crear un Cuenta de AWS, se empieza con una identidad de inicio de sesión que tiene acceso completo a todas Servicios de AWS y los recursos de la cuenta. Esta identidad se denomina Cuenta de AWS usuario root y se accede a él iniciando sesión con la dirección de correo electrónico y la contraseña que utilizó para crear la cuenta. Recomendamos encarecidamente que no utilice el

usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de tareas que requieren que inicie sesión como usuario root, consulte Tareas que requieren credenciales de usuario root en la Guía del IAM usuario.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos, incluidos los que requieren acceso de administrador, que utilicen la federación con un proveedor de identidades para acceder Servicios de AWS mediante credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidades web, el AWS Directory Service, el directorio del Centro de identidades o cualquier usuario que acceda Servicios de AWS mediante las credenciales proporcionadas a través de una fuente de identidad. Cuando las identidades federadas acceden Cuentas de AWS, asumen funciones y las funciones proporcionan credenciales temporales.

Para la administración centralizada del acceso, le recomendamos que utilice AWS IAM Identity Center. Puede crear usuarios y grupos en IAM Identity Center, o puede conectarse y sincronizarse con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todos sus Cuentas de AWS y aplicaciones. Para obtener información sobre IAM Identity Center, consulte ¿Qué es IAM Identity Center? en el AWS IAM Identity Center Guía del usuario.

Usuarios y grupos de IAM

Un IAMusuario es una identidad dentro de tu Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos utilizar credenciales temporales en lugar de crear IAM usuarios con credenciales de larga duración, como contraseñas y claves de acceso. Sin embargo, si tiene casos de uso específicos que requieren credenciales a largo plazo con IAM los usuarios, le recomendamos que rote las claves de acceso. Para obtener más información, consulte Rotar las claves de acceso con regularidad para los casos de uso que requieran credenciales de larga duración en la Guía del IAM usuario.

Un IAMgrupo es una identidad que especifica un conjunto de IAM usuarios. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos para grandes conjuntos de usuarios. Por ejemplo, puede asignar un nombre a un grupo IAMAdminsy concederle permisos para administrar IAM los recursos.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales de larga duración permanentes; no obstante, los roles proporcionan credenciales temporales. Para obtener más información, consulte Cuándo crear un IAM usuario (en lugar de un rol) en la Guía del IAM usuario.

IAMroles

Un IAMrol es una identidad dentro de tu Cuenta de AWS que tiene permisos específicos. Es similar a un IAM usuario, pero no está asociado a una persona específica. Puede asumir temporalmente un IAM rol en el AWS Management Console cambiando de rol. Puede asumir un rol llamando a un AWS CLI o AWS APIoperación o mediante una operación personalizadaURL. Para obtener más información sobre los métodos de uso de roles, consulte Uso de IAM roles en la Guía del IAM usuario.

IAMlos roles con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información sobre los roles para la federación, consulte Creación de un rol para un proveedor de identidad externo en la Guía del IAM usuario. Si usa IAM Identity Center, configura un conjunto de permisos. Para controlar a qué pueden acceder sus identidades después de autenticarse, IAM Identity Center correlaciona el conjunto de permisos con un rol en. IAM Para obtener información sobre los conjuntos de permisos, consulte los conjuntos de permisos en la AWS IAM Identity Center Guía del usuario.
- Permisos de IAM usuario temporales: un IAM usuario o rol puede asumir un IAM rol para asumir temporalmente diferentes permisos para una tarea específica.
- Acceso multicuenta: puedes usar un IAM rol para permitir que alguien (un responsable de confianza) de una cuenta diferente acceda a los recursos de tu cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunos Servicios de AWS, puede adjuntar una política directamente a un recurso (en lugar de utilizar un rol como proxy). Para saber la diferencia entre las funciones y las políticas basadas en recursos para el acceso multicuenta, consulta el tema sobre el acceso a los recursos entre cuentas IAM en la Guía del IAM usuario.
- Acceso entre servicios: algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realizas una llamada en un servicio, es habitual que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio

haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.

- Sesiones de acceso directo (FAS): cuando utilizas un IAM usuario o un rol para realizar acciones en AWS, se le considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FASutiliza los permisos del director que llama a un Servicio de AWS, combinado con la solicitud Servicio de AWS para realizar solicitudes a los servicios intermedios. FASlas solicitudes solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completar. En este caso, debe tener permisos para realizar ambas acciones. Para obtener detalles sobre la política a la hora de realizar FAS solicitudes, consulte Reenviar las sesiones de acceso.
- Función de servicio: una función de servicio es una IAMfunción que un servicio asume para realizar acciones en su nombre. Un IAM administrador puede crear, modificar y eliminar un rol de servicio desde dentroIAM. Para obtener más información, consulte Crear un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
- Función vinculada a un servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un Servicio de AWS. El servicio puede asumir la función de realizar una acción en su nombre. Los roles vinculados al servicio aparecen en su Cuenta de AWS y son propiedad del servicio. Un IAM administrador puede ver, pero no editar, los permisos de las funciones vinculadas al servicio.

- Aplicaciones que se ejecutan en Amazon EC2: puedes usar un IAM rol para administrar las credenciales temporales de las aplicaciones que se ejecutan en una EC2 instancia y se están creando AWS CLI o AWS APIsolicitudes. Esto es preferible a almacenar las claves de acceso dentro de la EC2 instancia. Para asignar un AWS Un rol a una EC2 instancia y ponerlo a disposición de todas sus aplicaciones, debe crear un perfil de instancia que se adjunte a la instancia. Un perfil de instancia contiene el rol y permite que los programas que se ejecutan en la EC2 instancia obtengan credenciales temporales. Para obtener más información, consulte Uso de un IAM rol para conceder permisos a aplicaciones que se ejecutan en EC2 instancias de Amazon en la Guía del IAM usuario.

Para saber si se deben usar IAM roles o IAM usuarios, consulte Cuándo crear un IAM rol (en lugar de un usuario) en la Guía del IAM usuario.

Administración de acceso mediante políticas

Usted controla el acceso en AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto en AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan en AWS como JSON documentos. Para obtener más información sobre la estructura y el contenido de los documentos de JSON políticas, consulte Descripción general de JSON las políticas en la Guía del IAM usuario.

Los administradores pueden utilizar AWS JSONpolíticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Para conceder a los usuarios permiso para realizar acciones en los recursos que necesitan, un IAM administrador puede crear IAM políticas. A continuación, el administrador puede añadir las IAM políticas a las funciones y los usuarios pueden asumir las funciones.

IAMlas políticas definen los permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre su función en AWS Management Console, el AWS CLI, o el AWS API.

Políticas basadas en identidad

Las políticas basadas en la identidad son documentos de política de JSON permisos que se pueden adjuntar a una identidad, como un IAM usuario, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener información sobre cómo crear una política basada en la identidad, consulte Creación de IAM políticas en la Guía del usuario. IAM

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y funciones de su Cuenta de AWS. Las políticas gestionadas incluyen AWS las políticas gestionadas y las políticas gestionadas por el cliente. Para saber cómo elegir entre una política gestionada o una política en línea, consulte Elegir entre políticas gestionadas y políticas integradas en la Guía del IAMusuario.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de JSON política que se adjuntan a un recurso. Algunos ejemplos de políticas basadas en recursos son las políticas de confianza de IAM roles y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o Servicios de AWS.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar AWS políticas gestionadas desde una política basada IAM en recursos.

Listas de control de acceso () ACLs

Las listas de control de acceso (ACLs) controlan qué responsables (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLsson similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de JSON políticas.

Amazon S3, AWS WAF, y Amazon VPC son ejemplos de servicios que admitenACLs. Para obtener más informaciónACLs, consulte la descripción general de la lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una función avanzada en la que se establecen los permisos máximos que una política basada en la identidad puede conceder a una IAM entidad (IAMusuario o rol). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte los límites de los permisos para IAM las entidades en la Guía del IAMusuario.

- Políticas de control de servicios (SCPs): SCPs son JSON políticas que especifican los permisos máximos para una organización o unidad organizativa (OU) en AWS Organizations. AWS Organizations es un servicio para agrupar y administrar de forma centralizada múltiples Cuentas de AWS que es propiedad de su empresa. Si habilitas todas las funciones de una organización, puedes aplicar las políticas de control de servicios (SCPs) a cualquiera de tus cuentas o a todas ellas. SCPLimita los permisos de las entidades en las cuentas de los miembros, incluidas todas Usuario raíz de la cuenta de AWS. Para obtener más información acerca de OrganizationsSCPs, consulte Políticas de control de servicios en AWS Organizations Guía del usuario.
- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información, consulte las políticas de sesión en la Guía del IAM usuario.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo AWS determina si se permite una solicitud cuando se trata de varios tipos de políticas, consulte la lógica de evaluación de políticas en la Guía del IAM usuario.

¿Cómo AWS HealthImaging funciona con IAM

Antes de administrar el IAM acceso a HealthImaging, infórmese sobre IAM las funciones disponibles para su uso HealthImaging.

IAMfunciones que puedes usar con AWS HealthImaging

IAMcaracterística	HealthImaging apoyo
Políticas basadas en identidades	Sí
Políticas basadas en recursos	No
Acciones de políticas	Sí

IAMcaracterística	HealthImaging apoyo
Recursos de políticas	Sí
Claves de condición de política (específicas del servicio)	Sí
ACLs	No
ABAC(etiquetas en las políticas)	Parcial
Credenciales temporales	Sí
Permisos de entidades principales	Sí
Roles de servicio	Sí
Roles vinculados al servicio	No

Para obtener una visión de alto nivel de cómo HealthImaging y otros AWS los servicios funcionan con la mayoría de IAM las funciones, consulte AWS servicios con los que funcionan IAM en la Guía IAM del usuario.

Políticas basadas en la identidad para HealthImaging

Compatibilidad con las políticas basadas en identidad: sí

Las políticas basadas en la identidad son documentos de política de JSON permisos que se pueden adjuntar a una identidad, como un IAM usuario, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener información sobre cómo crear una política basada en la identidad, consulte Creación de IAM políticas en la Guía del usuario. IAM

Con las políticas IAM basadas en la identidad, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. No es posible especificar la entidad principal en una política basada en identidad porque se aplica al usuario o rol al que está adjunto. Para obtener más información sobre todos los elementos que puede utilizar en una JSON política, consulte la referencia sobre los elementos de la IAM JSON política en la Guía del IAMusuario.

Ejemplos de políticas basadas en la identidad para HealthImaging

Para ver ejemplos de políticas HealthImaging basadas en la identidad, consulte. Ejemplos de políticas basadas en la identidad para AWS HealthImaging

Políticas basadas en recursos incluidas HealthImaging

Admite políticas basadas en recursos: no

Las políticas basadas en recursos son documentos de JSON política que se adjuntan a un recurso. Algunos ejemplos de políticas basadas en recursos son las políticas de confianza de IAM roles y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o Servicios de AWS.

Para habilitar el acceso entre cuentas, puede especificar una cuenta completa o IAM entidades de otra cuenta como principales en una política basada en recursos. Añadir a una política en función de recursos una entidad principal entre cuentas es solo una parte del establecimiento de una relación de confianza. Cuando el principal y el recurso son diferentes Cuentas de AWS, el IAM administrador de la cuenta de confianza también debe conceder permiso a la entidad principal (usuario o rol) para acceder al recurso. Para conceder el permiso, adjunte la entidad a una política basada en identidad. Sin embargo, si la política en función de recursos concede el acceso a una entidad principal de la misma cuenta, no es necesaria una política basada en identidad adicional. Para obtener más información, consulte el tema Acceso a recursos entre cuentas IAM en la Guía del IAM usuario.

Acciones políticas para HealthImaging

Compatibilidad con las acciones de política: sí

Los administradores pueden usar AWS JSONpolíticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El `Action` elemento de una JSON política describe las acciones que puede utilizar para permitir o denegar el acceso en una política. Las acciones políticas suelen tener el mismo nombre que las asociadas AWS APIoperación. Hay algunas excepciones, como las acciones que solo requieren permisos y que no tienen una operación coincidente. API También hay algunas operaciones que

requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de HealthImaging acciones, consulte las acciones definidas AWS HealthImaging en la Referencia de autorización de servicios.

Las acciones políticas HealthImaging utilizan el siguiente prefijo antes de la acción:

```
AWS
```

Para especificar varias acciones en una única instrucción, sepárelas con comas.

```
"Action": [
      "AWS:action1",
      "AWS:action2"
         ]
```

Para ver ejemplos de políticas HealthImaging basadas en la identidad, consulte. Ejemplos de políticas basadas en la identidad para AWS HealthImaging

Recursos de políticas para HealthImaging

Compatibilidad con los recursos de políticas: sí

Los administradores pueden usar AWS JSONpolíticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` JSON de política especifica el objeto o los objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso mediante su nombre de recurso de Amazon (ARN). Puede hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utilice un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

```
"Resource": "*"
```

Para ver una lista de los tipos de HealthImaging recursos y sus correspondientesARNs, consulte los tipos de recursos definidos AWS HealthImaging en la Referencia de autorización de servicios. Para saber con qué acciones y recursos puede utilizar unaARN, consulte Acciones definidas por AWS HealthImaging.

Para ver ejemplos de políticas HealthImaging basadas en la identidad, consulte. Ejemplos de políticas basadas en la identidad para AWS HealthImaging

Claves de condición de la política para HealthImaging

Compatibilidad con claves de condición de políticas específicas del servicio: sí

Los administradores pueden usar AWS JSONpolíticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puede crear expresiones condicionales que utilicen operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios `Condition` elementos en una declaración o varias claves en un solo `Condition` elemento, AWS los evalúa mediante una `AND` operación lógica. Si especifica varios valores para una sola clave de condición, AWS evalúa la condición mediante una `OR` operación lógica. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puede utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puede conceder a un IAM usuario permiso para acceder a un recurso solo si está etiquetado con su nombre de IAM usuario. Para obtener más información, consulte los elementos de IAM política: variables y etiquetas en la Guía del IAM usuario.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas AWS claves de condición globales, consulte AWS claves de contexto de condiciones globales en la Guía IAM del usuario.

Para ver una lista de claves de HealthImaging condición, consulte las claves de condición AWS HealthImaging en la Referencia de autorización de servicio. Para saber con qué acciones y recursos puede utilizar una clave de condición, consulte Acciones definidas por AWS HealthImaging.

Para ver ejemplos de políticas HealthImaging basadas en la identidad, consulte. Ejemplos de políticas basadas en la identidad para AWS HealthImaging

ACLsen HealthImaging

SoportesACLs: No

Las listas de control de acceso (ACLs) controlan qué directores (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLsson similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de JSON políticas.

RBACcon HealthImaging

Soportes RBAC	Sí

El modelo de autorización tradicional utilizado en se IAM denomina control de acceso basado en roles ()RBAC. RBACdefine los permisos en función de la función laboral de una persona, conocida fuera de AWS como un rol. Para obtener más información, consulte Comparación ABAC con el RBAC modelo tradicional en la Guía del IAM usuario.

ABACcon HealthImaging

Soportes ABAC (etiquetas en las políticas): parciales

> **Warning**
>
> ABACno se aplica mediante la `SearchImageSets` API acción. Cualquier persona que tenga acceso a la acción `SearchImageSets` puede acceder a todos los metadatos de los conjuntos de imágenes de un almacén de datos.

> **Note**
>
> Los conjuntos de imágenes son un recurso secundario de los almacenes de datos. Para poder utilizarloABAC, un conjunto de imágenes debe tener la misma etiqueta que un

banco de datos. Para obtener más información, consulte Etiquetar recursos con AWS HealthImaging.

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define los permisos en función de los atributos. En AWS, estos atributos se denominan etiquetas. Puede adjuntar etiquetas a IAM entidades (usuarios o roles) y a muchas AWS recursos. Etiquetar entidades y recursos es el primer paso deABAC. Luego, diseñe ABAC políticas para permitir las operaciones cuando la etiqueta del principal coincida con la etiqueta del recurso al que está intentando acceder.

ABACes útil en entornos de rápido crecimiento y ayuda en situaciones en las que la administración de políticas se vuelve engorrosa.

Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Si un servicio admite las tres claves de condición para cada tipo de recurso, el valor es Sí para el servicio. Si un servicio admite las tres claves de condición solo para algunos tipos de recursos, el valor es Parcial.

Para obtener más información al respectoABAC, consulte ¿Qué es? ABAC en la Guía IAM del usuario. Para ver un tutorial con los pasos de configuraciónABAC, consulte Usar el control de acceso basado en atributos (ABAC) en la Guía del IAMusuario.

Uso de credenciales temporales con HealthImaging

Compatibilidad con credenciales temporales: sí

Alguno Servicios de AWS no funcionan cuando inicias sesión con credenciales temporales. Para obtener información adicional, incluyendo qué Servicios de AWS trabaje con credenciales temporales, consulte Servicios de AWS que funcionan IAM en la Guía IAM del usuario.

Está utilizando credenciales temporales si inicia sesión en AWS Management Console utilizando cualquier método excepto un nombre de usuario y una contraseña. Por ejemplo, cuando accedes AWS mediante el enlace de inicio de sesión único (SSO) de su empresa, ese proceso crea automáticamente credenciales temporales. También crea credenciales temporales de forma automática cuando inicia sesión en la consola como usuario y luego cambia de rol. Para obtener más información sobre el cambio de rol, consulte Cambiar a un rol (consola) en la Guía del IAMusuario.

Puede crear credenciales temporales manualmente mediante el AWS CLI o AWS API. A continuación, puede utilizar esas credenciales temporales para acceder AWS. AWS recomienda generar credenciales temporales de forma dinámica en lugar de utilizar claves de acceso a largo plazo. Para obtener más información, consulte Credenciales de seguridad temporales en IAM.

Permisos principales entre servicios para HealthImaging

Admite sesiones de acceso directo (FAS): Sí

Cuando utiliza un IAM usuario o un rol para realizar acciones en AWS, se le considera director. Las políticas conceden permisos a una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. En este caso, debe tener permisos para realizar ambas acciones. Para ver si una acción requiere acciones dependientes adicionales en una política, consulte Acciones, recursos y claves de condición AWS HealthImaging en la Referencia de autorización de servicio.

Roles de servicio para HealthImaging

Compatibilidad con roles de servicio: sí

Una función de servicio es una IAMfunción que un servicio asume para realizar acciones en su nombre. Un IAM administrador puede crear, modificar y eliminar un rol de servicio desde dentroIAM. Para obtener más información, consulte Crear un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.

> **Warning**
>
> Cambiar los permisos de un rol de servicio podría interrumpir HealthImaging la funcionalidad. Edite las funciones de servicio solo cuando se HealthImaging proporcionen instrucciones para hacerlo.

Funciones vinculadas al servicio para HealthImaging

Compatibilidad con roles vinculados al servicio: no

Un rol vinculado a un servicio es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puede asumir la función de realizar una acción en su nombre. Los roles vinculados al servicio aparecen en su Cuenta de AWS y son propiedad del servicio. Un IAM administrador puede ver, pero no editar, los permisos de las funciones vinculadas al servicio.

Para obtener más información sobre la creación o la administración de funciones vinculadas a un servicio, consulte AWS servicios con los que funcionan. IAM Busque un servicio en la tabla que incluya `Yes` en la columna Rol vinculado a un servicio. Seleccione el vínculo Sí para ver la documentación acerca del rol vinculado a servicios para ese servicio.

Ejemplos de políticas basadas en la identidad para AWS HealthImaging

De forma predeterminada, los usuarios y los roles no tienen permiso para crear o modificar HealthImaging recursos. Tampoco pueden realizar tareas mediante el AWS Management Console, AWS Command Line Interface (AWS CLI), o AWS API. Para conceder a los usuarios permiso para realizar acciones en los recursos que necesitan, un IAM administrador puede crear IAM políticas. A continuación, el administrador puede añadir las IAM políticas a las funciones y los usuarios pueden asumir las funciones.

Para obtener información sobre cómo crear una política IAM basada en la identidad mediante estos documentos de JSON política de ejemplo, consulte Creación de IAM políticas en la Guía del IAMusuario.

Para obtener más información sobre las acciones y los tipos de recursos definidos por Awesome, incluido el formato de cada uno de los tipos de recursos, consulte Acciones, recursos y claves de condición ARNs para AWS Impresionante en la referencia de autorización de servicio.

Temas

- Prácticas recomendadas sobre las políticas
- Uso de la HealthImaging consola
- Cómo permitir a los usuarios consultar sus propios permisos

Prácticas recomendadas sobre las políticas

Las políticas basadas en la identidad determinan si alguien puede crear HealthImaging recursos de tu cuenta, acceder a ellos o eliminarlos. Estas acciones pueden suponer costes para su Cuenta de AWS. Al crear o editar políticas basadas en la identidad, siga estas directrices y recomendaciones:

- Comience con AWS políticas gestionadas y avance hacia los permisos con los privilegios mínimos: para empezar a conceder permisos a sus usuarios y cargas de trabajo, utilice la AWS políticas gestionadas que conceden permisos para muchos casos de uso habituales. Están disponibles en su Cuenta de AWS. Le recomendamos que reduzca aún más los permisos definiendo AWS políticas gestionadas por el cliente que sean específicas para sus casos de uso. Para obtener más

información, consulte AWS políticas gestionadas o AWS políticas gestionadas para las funciones laborales en la Guía IAM del usuario.

- Aplique permisos con privilegios mínimos: cuando establezca permisos con IAM políticas, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Para obtener más información sobre cómo IAM aplicar permisos, consulte Políticas y permisos IAM en la IAM Guía del usuario.
- Utilice las condiciones en IAM las políticas para restringir aún más el acceso: puede añadir una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puede escribir una condición de política para especificar que todas las solicitudes deben enviarse medianteSSL. También puede utilizar condiciones para conceder el acceso a las acciones del servicio si se utilizan a través de un procedimiento específico Servicio de AWS, como, por ejemplo, AWS CloudFormation. Para obtener más información, consulte los elementos IAM JSON de la política: Condición en la Guía del IAM usuario.
- Utilice IAM Access Analyzer para validar sus IAM políticas y garantizar permisos seguros y funcionales: IAM Access Analyzer valida las políticas nuevas y existentes para que se ajusten al lenguaje de las políticas (JSON) y IAM a las IAM mejores prácticas. IAMAccess Analyzer proporciona más de 100 comprobaciones de políticas y recomendaciones prácticas para ayudarle a crear políticas seguras y funcionales. Para obtener más información, consulte la validación de políticas de IAM Access Analyzer en la Guía del IAM usuario.
- Requerir autenticación multifactorial (MFA): si tiene un escenario que requiere IAM usuarios o un usuario raíz en su Cuenta de AWS, actívala MFA para mayor seguridad. Para solicitarlo MFA cuando se cancelen API las operaciones, añada MFA condiciones a sus políticas. Para obtener más información, consulte Configuración del API acceso MFA protegido en la Guía del IAM usuario.

Para obtener más información sobre las prácticas recomendadasIAM, consulte las prácticas recomendadas de seguridad IAM en la Guía del IAM usuario.

Uso de la HealthImaging consola

Para acceder a la AWS HealthImaging consola, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle enumerar y ver detalles sobre los HealthImaging recursos de su Cuenta de AWS. Si creas una política basada en la identidad que sea más restrictiva que los permisos mínimos requeridos, la consola no funcionará según lo previsto para las entidades (usuarios o roles) que cuenten con esa política.

No es necesario conceder permisos mínimos de consola a los usuarios que solo realizan llamadas al AWS CLI o el AWS API. En su lugar, permita el acceso únicamente a las acciones que coincidan con la API operación que están intentando realizar.

Para garantizar que los usuarios y los roles puedan seguir utilizando la HealthImaging consola, adjunte también la opción HealthImaging *ConsoleAccess* o *ReadOnly* AWS política gestionada a las entidades. Para obtener más información, consulte Añadir permisos a un usuario en la Guía del IAM usuario.

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo se muestra cómo se puede crear una política que permita a IAM los usuarios ver las políticas integradas y administradas asociadas a su identidad de usuario. Esta política incluye permisos para completar esta acción en la consola o mediante programación mediante el AWS CLI o AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
```

```
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Políticas administradas por AWS para AWS HealthImaging

Una política administrada de AWS es una política independiente que AWS crea y administra. Las políticas administradas de AWS se diseñan para ofrecer permisos para muchos casos de uso comunes, por lo que puede empezar a asignar permisos a los usuarios, grupos y roles.

Tenga presente que es posible que las políticas administradas de AWS no concedan permisos de privilegio mínimo para los casos de uso concretos, ya que están disponibles para que las utilicen todos los clientes de AWS. Se recomienda definir políticas administradas por el cliente para los casos de uso a fin de reducir aún más los permisos.

No puede cambiar los permisos definidos en las políticas administradas por AWS. Si AWS actualiza los permisos definidos en una política administrada de AWS, la actualización afecta a todas las identidades de entidades principales (usuarios, grupos y roles) a las que está adjunta la política. Lo más probable es que AWS actualice una política administrada de AWS cuando se lance un nuevo Servicio de AWS o las operaciones de la API nuevas estén disponibles para los servicios existentes.

Para obtener más información, consulte Políticas administradas de AWS en la Guía del usuario de IAM.

Temas

- Política administrada por AWS: AWSHealthImagingFullAccess
- Política administrada por AWS: AWSHealthImagingReadOnlyAccess
- Actualizaciones de HealthImaging en las políticas administradas de AWS

Política administrada por AWS: AWSHealthImagingFullAccess

Puede adjuntar la política `AWSHealthImagingFullAccess` a las identidades de IAM.

Esta política concede permisos administrativos a todas las acciones de HealthImaging.

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

Política administrada por AWS: AWSHealthImagingReadOnlyAccess

Puede adjuntar la política `AWSHealthImagingReadOnlyAccess` a las identidades de IAM.

Esta política concede permisos de solo lectura a acciones específicas de AWS HealthImaging.

```
{
    "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": "*"
    }]
}
```

Actualizaciones de HealthImaging en las políticas administradas de AWS

Es posible consultar los detalles sobre las actualizaciones de HealthImaging en las políticas administradas de AWS debido a que HealthImaging comenzó a realizar el seguimiento de estos cambios. Para obtener alertas automáticas sobre cambios en esta página, suscríbase a la fuente RSS en la página de Versiones.

Cambio	Descripción	Fecha
HealthImaging comenzó a realizar un seguimiento de los cambios	HealthImaging comenzó a realizar un seguimiento de los cambios de las políticas administradas de AWS.	19 de julio de 2023

Solución de problemas de identidad y acceso en AWS HealthImaging

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas más comunes que pueden surgir al trabajar con HealthImaging y. IAM

Temas

- No estoy autorizado a realizar ninguna acción en HealthImaging
- No estoy autorizado a realizar tareas como: PassRole
- Quiero permitir que personas ajenas a mi Cuenta de AWS para acceder a mis HealthImaging recursos

No estoy autorizado a realizar ninguna acción en HealthImaging

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

El siguiente ejemplo de error se produce cuando el `mateojackson` IAM usuario intenta usar la consola para ver los detalles de un *`my-example-widget`* recurso ficticio, pero no tiene los `AWS:`*`GetWidget`* permisos ficticios.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 AWS:GetWidget on resource: my-example-widget
```

En este caso, la política del usuario `mateojackson` debe actualizarse para permitir el acceso al recurso *`my-example-widget`* mediante la acción `AWS:`*`GetWidget`*.

Si necesitas ayuda, ponte en contacto con tu AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

No estoy autorizado a realizar tareas como: PassRole

Si recibes un mensaje de error que indica que no estás autorizado a realizar la `iam:PassRole` acción, debes actualizar tus políticas para que puedas transferirle HealthImaging una función.

Alguno Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir el rol al servicio.

El siguiente ejemplo de error se produce cuando un IAM usuario denominado `marymajor` intenta utilizar la consola para realizar una acción en ella. HealthImaging Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir que personas ajenas a mi Cuenta de AWS para acceder a mis HealthImaging recursos

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admiten políticas basadas en recursos o listas de control de acceso (ACLs), puedes usar esas políticas para permitir que las personas accedan a tus recursos.

Para más información, consulte lo siguiente:

- Para saber si HealthImaging es compatible con estas funciones, consulte. ¿Cómo AWS HealthImaging funciona con IAM
- Para obtener información sobre cómo proporcionar acceso a sus recursos en todas partes Cuentas de AWS que te pertenezca, consulta Proporcionar acceso a un IAM usuario en otro Cuenta de AWS que le pertenezca en la Guía IAM del usuario.
- Para obtener información sobre cómo proporcionar acceso a sus recursos a terceros Cuentas de AWS, consulte Proporcionar acceso a Cuentas de AWS propiedad de terceros en la Guía IAM del usuario.
- Para obtener información sobre cómo proporcionar acceso mediante la federación de identidades, consulte Proporcionar acceso a usuarios autenticados externamente (federación de identidades) en la Guía del IAM usuario.
- Para saber la diferencia entre el uso de roles y políticas basadas en recursos para el acceso entre cuentas, consulte el acceso a recursos entre cuentas IAM en la Guía del usuario. IAM

Validación de la conformidad de AWS HealthImaging

Auditores externos evalúan la seguridad y la conformidad de AWS HealthImaging en numerosos programas de conformidad de AWS. En el caso de HealthImaging, esto incluye la HIPAA.

Para obtener una lista de servicios de AWS en el ámbito de programas de conformidad específicos, consulte Servicios de AWS en el ámbito del programa de conformidad. Para obtener información general, consulte Programas de conformidad de AWS.

Puede descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

Su responsabilidad en el ámbito de la conformidad al usar AWS HealthImaging viene determinada por la confidencialidad de los datos, los objetivos de conformidad de su empresa y las leyes y regulaciones aplicables. AWS proporciona los siguientes recursos para ayudarlo con los requisitos de conformidad:

- Soluciones de socios de AWS: estas guías de seguridad e implementación tratan de la arquitectura y ofrecen pasos para implementar los entornos de referencia centrados en la seguridad y la conformidad en AWS.
- Documento técnico sobre arquitectura para seguridad y conformidad de HIPAA : en este documento técnico, se describe cómo las empresas pueden utilizar AWS para crear aplicaciones conformes con HIPAA.
- GxP Systems on AWS: este documento técnico proporciona información sobre cómo aborda AWS el cumplimiento y la seguridad relacionados con la práctica recomendada y proporciona orientación sobre el uso de los servicios de AWS en el contexto de la práctica recomendada.
- AWSRecursos de conformidad: este conjunto de manuales y guías podría aplicarse a su sector y ubicación.
- Evaluación de recursos con reglas: AWS Config determina en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normativas.
- AWS Security Hub: este servicio de AWS proporciona una vista integral de su estado de seguridad en AWS que lo ayuda a verificar la conformidad con los estándares y las prácticas recomendadas del sector de seguridad.

Seguridad de infraestructuras en AWS HealthImaging

Al tratarse de un servicio administrado, AWS HealthImaging está protegido por los procedimientos de seguridad de red globales de AWS, que se describen en el documento técnico Amazon Web Services: Información general sobre procesos de seguridad.

Las llamadas a la API publicadas en AWS se utilizan para obtener acceso a HealthImaging a través de la red. Los clientes deben ser compatibles con la seguridad de la capa de transporte (TLS) 1.3 o

posterior. Los clientes también deben ser compatibles con conjuntos de cifrado con confidencialidad directa total (PFS) tales como Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Creación de recursos de AWS HealthImaging con AWS CloudFormation

AWS HealthImaging está integrado con AWS CloudFormation, un servicio que le ayuda a modelar y configurar sus recursos de AWS para que pueda dedicar menos tiempo a crear y administrar sus recursos e infraestructura. Puede crear una plantilla que describa todos los recursos de AWS que desea y AWS CloudFormation los aprovisionará y configurará en su lugar.

Con AWS CloudFormation, puede volver reutilizar la plantilla para configurar sus recursos de HealthImaging varias veces, siempre manteniendo la coherencia. Solo tiene que describir los recursos una vez y luego aprovisionar los mismos recursos una y otra vez en varias Cuentas de AWS y regiones.

HealthImaging y plantillas de AWS CloudFormation

Para aprovisionar y configurar los recursos de HealthImaging y otros servicios relacionados, debe entender las plantillas de AWS CloudFormation. Las plantillas son archivos de texto con formato de tipo JSON o YAML. Estas plantillas describen los recursos que desea aprovisionar en sus pilas de AWS CloudFormation. Si no está familiarizado con JSON o YAML, puede utilizar Designer de AWS CloudFormation para comenzar a utilizar las plantillas de AWS CloudFormation. Para obtener más información, consulte ¿Qué es Designer de AWS CloudFormation? en la Guía del usuario de AWS CloudFormation.

AWS HealthImaging admite la creación de almacenes de datos con AWS CloudFormation. Para obtener más información, incluidos ejemplos de plantillas JSON y YAML para alimentar los almacenes de HealthImaging, consulte la referencia del tipo de recurso de AWS HealthImaging en la guía del usuario de AWS CloudFormation.

Obtener más información sobre AWS CloudFormation

Para obtener más información acerca de AWS CloudFormation, consulte los siguientes recursos:

- AWS CloudFormation
- Guía del usuario de AWS CloudFormation
- Referencia de la API de AWS CloudFormation
- Guía del usuario de la interfaz de la línea de comandos de AWS CloudFormation

AWS HealthImaging y puntos finales de VPC de interfaz ()AWS PrivateLink

Puede establecer una conexión privada entre su VPC y crear un punto final de AWS HealthImaging la VPC de interfaz. Los puntos de enlace de la interfaz funcionan con una tecnología que puede usar para acceder de forma privada a HealthImaging las API sin una puerta de enlace a Internet, un dispositivo NAT, una conexión VPN o una conexión AWS Direct Connect. AWS PrivateLink Las instancias de su VPC no necesitan direcciones IP públicas para comunicarse con HealthImaging las API. El tráfico entre tu VPC y HealthImaging no sale de la red de Amazon.

Cada punto de conexión de la interfaz está representado por una o más interfaces de red elásticas en las subredes.

Para obtener más información, consulte Interface VPC Endpoints (AWS PrivateLink) en la Guía del usuario de Amazon VPC.

Temas

- Consideraciones sobre los puntos HealthImaging finales de VPC
- Creación de un punto final de VPC de interfaz para HealthImaging
- Crear una política de puntos de conexión de VPC para HealthImaging

Consideraciones sobre los puntos HealthImaging finales de VPC

Antes de configurar un punto de enlace de VPC de interfaz HealthImaging, asegúrese de revisar las propiedades y limitaciones del punto de enlace de interfaz en la Guía del usuario de Amazon VPC.

HealthImaging permite realizar llamadas a todas las AWS HealthImaging acciones desde su VPC.

Creación de un punto final de VPC de interfaz para HealthImaging

Puede crear un punto de enlace de VPC para el HealthImaging servicio mediante la consola de Amazon VPC o el (). AWS Command Line Interface AWS CLI Para más información, consulte Creación de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Cree puntos de enlace de VPC para HealthImaging usar los siguientes nombres de servicio:

- com.amazonaws.*region*.medical-imaging
- com.amazonaws. *región.* runtime-medical-imaging
- com.amazonaws. *región.* dicom-medical-imaging

> **Note**
>
> El DNS privado debe estar habilitado para su uso PrivateLink.

Puede realizar solicitudes a la API para HealthImaging utilizar su nombre de DNS predeterminado para la región, por ejemplo,`medical-imaging.us-east-1.amazonaws.com`.

Para más información, consulte Acceso a un servicio a través de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Crear una política de puntos de conexión de VPC para HealthImaging

Puede adjuntar una política de punto final a su punto final de VPC que controle el acceso a. HealthImaging La política especifica la siguiente información:

- La entidad principal que puede realizar acciones
- Las acciones que se pueden realizar
- Los recursos en los que se pueden llevar a cabo las acciones

Para más información, consulte Control del acceso a los servicios con puntos de enlace de la VPC en la Guía del usuario de Amazon VPC.

Ejemplo: política de puntos finales de VPC para acciones HealthImaging

El siguiente es un ejemplo de una política de puntos finales para HealthImaging. Cuando se adjunta a un punto final, esta política otorga acceso a HealthImaging las acciones a todos los principales de todos los 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\":\"*\"}]}"
```

Importación multicuenta para AWS HealthImaging

Con la importación entre cuentas o regiones, puede importar datos a su HealthImaging almacén de datos desde buckets de Amazon S3 ubicados en otras regiones compatibles. Puede importar datos entre AWS cuentas, cuentas que sean propiedad de otras AWS Organizaciones y desde fuentes de datos abiertas, como Imaging Data Commons (IDC), que se encuentran en el Registro de Datos Abiertos de. AWS

HealthImaging Los casos de uso de la importación entre cuentas y regiones incluyen:

- Productos SaaS de imágenes médicas que importan datos DICOM de cuentas de clientes

- Grandes organizaciones que llenan un almacén de HealthImaging datos a partir de varios depósitos de entrada de Amazon S3
- Los investigadores comparten datos de forma segura en estudios clínicos realizados en varias instituciones

Para utilizar la importación multicuenta

1. El propietario del bucket de entrada (fuente) de Amazon S3 debe conceder los `s3:GetObject` permisos `s3:ListBucket` y el propietario del almacén de HealthImaging datos.
2. El propietario del almacén de HealthImaging datos debe añadir el bucket de Amazon S3 a su IAM. `ImportJobDataAccessRole` Consulte Cree un IAM rol para la importación.
3. El propietario del almacén de HealthImaging datos debe proporcionar `inputOwnerAccountId`el depósito de entrada de Amazon S3 al iniciar el trabajo de importación.

> **Note**
>
> Al proporcionarla`inputOwnerAccountId`, el propietario del almacén de datos valida la entrada que el bucket de Amazon S3 pertenece a la cuenta especificada para mantener el cumplimiento de los estándares del sector y mitigar los posibles riesgos de seguridad.

El siguiente ejemplo de `startDICOMImportJob` código incluye el `inputOwnerAccountId` parámetro opcional, que se puede aplicar a todos los AWS CLI ejemplos de código del SDK de la Inicio de un trabajo de importación sección.

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 "";
}
```

Resiliencia en AWS HealthImaging

La infraestructura global de AWS se divide en Regiones de AWS y zonas de disponibilidad. Las Regiones de AWS proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre las Regiones de AWS y las zonas de disponibilidad, consulte Infraestructura global de AWS.

Además de la infraestructura global de AWS, AWS HealthImaging ofrece varias características que le ayudan con sus necesidades de resiliencia y copia de seguridad de los datos.

Material de HealthImaging referencia de AWS

El siguiente material de referencia está disponible para AWS HealthImaging.

> **Note**
>
> Todas HealthImaging las acciones y tipos de datos se encuentran en una referencia independiente. Para obtener más información, consulte la referencia de HealthImaging API de AWS.

Temas

DICOMsoporte para AWS HealthImaging

AWS HealthImaging admite DICOM elementos específicos y sintaxis de transferencia. Familiarícese con los elementos de DICOM datos compatibles a nivel de paciente, estudio y serie, ya que las claves de HealthImaging metadatos se basan en ellos. Antes de iniciar una importación, compruebe que sus datos de imágenes médicas cumplen con las sintaxis HealthImaging de transferencia y las restricciones de DICOM elementos compatibles.

> **Note**
>
> AWS HealthImaging actualmente no admite imágenes de segmentación binaria ni datos de píxeles de secuencias de imágenes de iconos.

Temas

Clases compatibles SOP

Con él AWS HealthImaging, puedes importar instancias DICOM P10 Service-Object Pair (SOP) codificadas con cualquier SOP claseUID, incluidas las retiradas y las privadas. También se conservan todos los atributos privados.

Normalización de metadatos

Al importar los datos del DICOM P10 AWS HealthImaging, estos se transforman en conjuntos de imágenes compuestos por metadatos y marcos de imágenes (datos en píxeles). Durante el proceso de transformación, las claves de HealthImaging metadatos se generan en función de una versión específica del DICOM estándar. HealthImaging actualmente genera y admite claves de metadatos basadas en el diccionario de datos DICOM PS3 .6 2022b.

AWS HealthImaging admite los siguientes elementos DICOM de datos a nivel de paciente, estudio y serie.

Elementos a nivel de paciente

> **Note**
>
> Para obtener una descripción detallada de cada elemento a nivel de paciente, consulte el Registro de elementos de DICOM datos.

AWS HealthImaging admite los siguientes elementos a nivel 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 a nivel de estudio

> **Note**
>
> Para obtener una descripción detallada de cada elemento del nivel de estudio, consulte el Registro de elementos de DICOM datos.

AWS HealthImaging admite los siguientes elementos del nivel de estudio:

```
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 a nivel de serie

> **Note**
>
> Para obtener una descripción detallada de cada elemento de nivel de serie, consulte el Registro de elementos de DICOM datos.

AWS HealthImaging admite los siguientes elementos de nivel de serie:

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
```

Sintaxis de transferencia compatibles

AWS HealthImaging importa SOP instancias DICOM P10 codificadas con las sintaxis de transferencia que se muestran en la siguiente tabla. Además del almacenamiento de la SOP instancia, HealthImaging transcodifica los fotogramas de imagen (datos de píxeles) HTJ2K para convertirlos en SOP instancias codificadas con las siguientes sintaxis de transferencia:

Sintaxis de transferencia UID	Nombre de la sintaxis de transferencia
1.2.840.10008.1.2	Endian VR implícito: sintaxis de transferencia predeterminada para DICOM
1.2.840.10008.1.2.1	VR Little Endian explícita
1.2,840,10008,12.1,99	VR Little Endian deflated explicita
1.2,840,10008.1.2.2	VR Big Endian explícita
1.2,840,100008,12,4,50	JPEGBase de referencia (proceso 1): sintaxis de transferencia predeterminada para la compresión de imágenes de 8 bits con pérdidas JPEG
1.2.840.10008.1.2.4.51	JPEGBase de referencia (procesos 2 y 4): sintaxis de transferencia predeterminada para la compresión de imágenes de 12 bits con pérdidas (solo en el proceso 4) JPEG

Sintaxis de transferencia UID	Nombre de la sintaxis de transferencia
1.2.840.10008.1.2.4.57	JPEGSin pérdidas y no jerárquico (proceso 14)
1.2.840.10008.1.2.4.70	JPEGPredicción sin pérdidas, no jerárquic a y de primer orden (procesos 14 [valor de selección 1]): sintaxis de transferencia predeterminada para la compresión de imágenes sin pérdidas JPEG
1.2.840.10008.1.2.4.80	JPEG-LS Compresión de imagen sin pérdidas
1.2.840.10008.1.2.4.81	JPEG-LS Compresión de imagen con pérdida (casi sin pérdidas)
1.2.840.10008.1.2.4.90	JPEGCompresión de imágenes en 2000 (solo sin pérdidas)
1.2.840.10008.1.2.4.91	JPEGCompresión de imágenes en 2000
1.2.840.10008.1.2.4.201	Compresión de imágenes 2000 de alto rendimiento (solo sin pérdidas) JPEG
1.2.840.10008.1.2.4.202	2000 de alto rendimiento con opciones de compresión de imágenes JPEG (solo sin pérdidas) RPCL
1.2.840.10008.1.2.4.203	Compresión de imágenes 2000 de alto rendimiento JPEG
1.2.840.10008.1.2.5	RLESin pérdidas

DICOMrestricciones de elementos

Al importar sus datos de imágenes médicas AWS HealthImaging, se aplican restricciones de longitud máxima a los siguientes DICOM elementos. Para que la importación sea correcta, asegúrese de que sus datos no superen las restricciones de longitud máxima.

DICOMrestricciones de elementos durante la importación

HealthImaging palabra clave	Palabra clave DICOM	clave DICOM	Límite de longitud
DICOMPatientName	Nombre del paciente	(0010,0010)	mínimo: 0, máximo: 256
DICOMPatientID	ID de paciente	(0010,0020)	mínimo: 0, máximo: 256
DICOMPatientBirthDate	Paciente BirthDate	(0010,0030)	mínimo: 0, máximo: 18
DICOMPatientSex	PatientSex	(0010,0040)	mínimo: 0, máximo: 16
DICOMStudyInstanceUID	StudyInstanceUID	(0020,000D)	mínimo: 0, máximo: 256
DICOMStudyId	StudyID	(0020,0010)	mínimo: 0, máximo: 16
DICOMStudyDescription	StudyDescription	(0008,1030)	mínimo: 0, máximo: 64
DICOMNumberOfStudy RelatedSeries	NumberOfStudyRelatedSeries	(0020,1206)	mín: 0, máximo: 1.000.000
DICOMNumberOfStudy RelatedInstances	NumberOfStudyRelatedInstancias	(0020,1208)	mínimo: 0, máximo: 10.000
DICOMAccessionNumber	AccessionNumber	(0008,0050)	mínimo: 0, máximo: 256
DICOMStudyDate	StudyDate	(0008,0020)	mínimo: 0, máximo: 18

HealthImaging palabra clave	Palabra clave DICOM	clave DICOM	Límite de longitud
DICOMStudyTime	StudyTime	(0008,0030)	mínimo: 0, máximo: 28

DICOMrestricciones de metadatos

Cuando se utilizan `UpdateImageSetMetadata` para actualizar HealthImaging los atributos de los metadatos, se aplican DICOM las siguientes restricciones.

- No se pueden actualizar ni eliminar los atributos privados de los atributos a nivel de paciente, estudio, serie o instancia, a menos que la restricción de actualización se aplique tanto a `updatableAttributes` `removableAttributes`
- No se pueden actualizar los siguientes atributos AWS HealthImaging generados:`SchemaVersion`,,,,,,, `DatastoreID` `ImageSetID` `PixelData` `Checksum` `Width` `Height` `MinPixelValue` `MaxPixelValue` `FrameSizeInBytes`
- No se pueden actualizar los siguientes DICOM atributos a menos que el `force` indicador esté activado:`Tag.PixelData`,`Tag.StudyInstanceUID`,,`Tag.SeriesInstanceUID`,`Tag.SOPInstanceU` `Tag.StudyID`
- No se pueden actualizar los atributos con el tipo VR SQ (atributos anidados) a menos que el `force` indicador esté activado
- No se pueden actualizar los atributos multivalor a menos que el indicador esté activado `force`
- No se pueden actualizar los atributos con valores que no sean compatibles con el tipo VR del atributo a menos que el `force` indicador esté activado
- No se pueden actualizar los atributos que no se consideran válidos según el DICOM estándar, a menos que se `force` active el indicador
- No se pueden actualizar los atributos de todos los módulos. Por ejemplo, si se proporciona un atributo a nivel de paciente a nivel de estudio en la solicitud de carga útil del cliente, la solicitud puede invalidarse.
- No se pueden actualizar los atributos si el módulo de atributos asociado no está presente en los `ImageSetMetadata` existentes. Por ejemplo, no está permitido actualizar los atributos de un `seriesInstanceUID` si la serie con `seriesInstanceUID` no está presente en los metadatos del conjunto de imágenes existente.

Verificación de datos de HealthImaging píxeles de AWS

Durante la importación, HealthImaging proporciona una verificación integrada de los datos de píxeles al comprobar el estado de codificación y decodificación sin pérdidas de cada imagen. Esta función garantiza que las imágenes decodificadas mediante las bibliotecas de decodificación HTJ2K siempre coincidan con las imágenes DICOM P10 originales importadas. HealthImaging

- El proceso de incorporación de imágenes comienza cuando un trabajo de importación captura el estado original de calidad de píxeles de las imágenes DICOM P10 antes de importarlas. Mediante el algoritmo CRC32, se genera una suma de verificación de la resolución de fotogramas (IFRC) única e inmutable para cada imagen. Se calcula una IFRC por nivel de resolución para los datos de píxeles de cada imagen. Los valores de la suma de verificación de la IFRC se presentan en el documento de metadatos (`job-output-manifest.json`), ordenados en una lista desde la resolución básica hasta la resolución completa.
- Una vez importadas las imágenes a un almacén de HealthImaging datos y transformadas en conjuntos de imágenes, los marcos de imágenes codificados en HTJ2K se decodifican inmediatamente y se calculan las nuevas IFRC. HealthImaging a continuación, compara las IFRC de resolución completa de las imágenes originales con las nuevas IFRC de los marcos de imágenes importados para comprobar su precisión.
- La condición de error descriptiva correspondiente por imagen se captura en el registro de resultados del trabajo de importación (`job-output-manifest.json`) para que pueda revisarla y verificarla.

Cómo verificar los datos de píxeles

1. Tras importar los datos de imágenes médicas, consulte la descripción correcta (o la condición de error) de cada conjunto de imágenes capturada en el registro de resultados del trabajo de importación, `job-output-manifest.json`. Para obtener más información, consulte Introducción a los trabajos de importación.
2. Los conjuntos de imágenes se componen de metadatos y marcos de imágenes (datos en píxeles). Los metadatos del conjunto de imágenes contienen información sobre los marcos de imágenes asociados. Utilice la `GetImageSetMetadata` acción para obtener los metadatos de un conjunto de imágenes. Para obtener más información, consulte Obtención de metadatos de conjuntos de imágenes.
3. `PixelDataChecksumFromBaseToFullResolution` contiene la IFRC (suma de verificación) por nivel de resolución. A continuación se muestra un ejemplo de salida de metadatos para la

IFRC que se genera como parte del proceso de importación y se graba en ella. `job-output-manifest.json`

```
"ImageFrames": [{
  "ID": "678906789067890123451234512345 12",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 128,
      "Height": 128,
      "Checksum": 2928338830
    },
    {
      "Width": 256,
      "Height": 256,
      "Checksum": 1362274918
    },
    {
      "Width": 512,
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
```

4. Para verificar los datos de píxeles, acceda al procedimiento de verificación de datos de píxeles GitHub y siga las instrucciones del `README.md` archivo para verificar de forma independiente el procesamiento de imágenes sin pérdidas por parte de los distintos procesadores Bibliotecas de decodificación HTJ2K que utilizan. HealthImaging A medida que los datos se cargan progresivamente según el nivel de resolución, usted puede calcular la IFRC para los datos de entrada sin procesar y compararlos con el valor de la IFRC proporcionado en los HealthImaging metadatos para esa misma resolución a fin de verificar los datos de píxeles.

Bibliotecas de decodificación HTJ2K para AWS HealthImaging

Durante la importación, AWS HealthImaging codifica todos los marcos de imagen (datos de píxeles) en el formato JPEG 2000 (HTJ2K) de alto rendimiento sin pérdidas para ofrecer una visualización de imágenes rápida y uniforme y un acceso universal a las funciones avanzadas de HTJ2K. Como los marcos de imagen se codifican en HTJ2K durante la importación, deben decodificarse antes de visualizarlos en un visor de imágenes.

> **Note**
>
> El HTJ2K se define en la parte 15 de la norma JPEG2000 (ISO/IEC 15444-15:2019). HTJ2K conserva las funciones avanzadas de JPEG2000, como la escalabilidad de la resolución, los precintos, el mosaico, la alta profundidad de bits, los canales múltiples y la compatibilidad con el espacio de color.

Temas

Bibliotecas de decodificación HTJ2K

Según el lenguaje de programación que utilice, le recomendamos las siguientes bibliotecas de decodificación para decodificar marcos de imágenes.

- NVIDIA NVJPEG2000: comercial, acelerada por GPU
- Software Kakadu: comercial, C++ con enlaces a Java y .NET
- OpenJPH: código abierto, C++ y WASM
- OpenJPEG: código abierto, C/C++, Java
- openjphpy: código abierto, Python
- pylibjpeg-openjpeg: código abierto, Python

Visores de imágenes

Puede ver los marcos de las imágenes después de decodificarlos. Las acciones HealthImaging de la API de AWS admiten diversos visores de imágenes de código abierto, entre los que se incluyen:

- Open Health Imaging Foundation (OHIF)
- Cornerstone.js

AWS HealthImaging puntos finales y cuotas

Los siguientes temas contienen información sobre los puntos finales y las cuotas del AWS HealthImaging servicio.

Temas

Puntos de conexión de servicio

Un punto final de servicio es URL aquel que identifica un host y un puerto como punto de entrada para un servicio web. Cada solicitud de servicio web contiene un punto de enlace. La mayoría AWS de los servicios proporcionan puntos finales para regiones específicas a fin de permitir una conectividad más rápida. En la siguiente tabla se enumeran los puntos finales del servicio para. AWS HealthImaging

Nombre de la región	Región	Punto de conexión	Protocolo	
Este de EE. UU. (Norte de Virginia)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS	
Oeste de EE. UU. (Oregón)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS	
Asia-Pacífico (Sídney)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS	
Europa (Irlanda)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS	

Si utiliza HTTP solicitudes para llamar a AWS HealthImaging acciones, debe usar diferentes puntos de enlace en función de las acciones a las que se invoque. El siguiente menú muestra los puntos finales de servicio disponibles para HTTP las solicitudes y las acciones que admiten.

APIAcciones compatibles para las solicitudes HTTP

data store, import, tagging

Se puede acceder a las siguientes acciones de almacenamiento, importación y etiquetado de datos desde el punto final:

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- Un tartDICOMImport trabajo
- G etDICOMImport Job
- istDICOMImportEmpleos L
- TagResource
- ListTagsForResource
- UntagResource

image set

Se puede acceder a las siguientes acciones del conjunto de imágenes desde el punto final:

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging ofrece una representación de los servicios DICOMweb Retrieve WADO -RS. Para obtener más información, consulte Recuperación de DICOM datos de HealthImaging.

Se puede acceder a los siguientes DICOMweb servicios a través de un punto final:

```
https://dicom-medical-imaging.region.amazonaws.com
```

- `GetDICOMInstance`
- `GetDICOMInstanceMetadata`
- `GetDICOMInstanceFrames`

Service Quotas

Las cuotas de servicio se definen como el valor máximo de los recursos, acciones y elementos de su AWS cuenta.

> **Note**
>
> Puede solicitar aumentos de cuota para cuotas ajustables mediante Service Quotas. Para obtener más información, consulte Solicitud de un aumento de cuota en la Guía del usuario de Service Quotas.

En la siguiente tabla se muestran las cuotas predeterminadas de AWS HealthImaging.

Nombre	Valor predeterminado	Ajusta	Descripción
Número máximo de CopyImageSet solicitudes simultáneas por almacén de datos	Cada región admitida: 100	Sí	El número máximo de CopyImageSet solicitudes simultáneas por almacén de datos en la región actual AWS
Número máximo de DeleteImageSet solicitudes simultáneas por almacén de datos	Cada región admitida: 100	Sí	El número máximo de DeleteImageSet solicitudes simultáneas por almacén de datos en la región actual AWS
Número máximo de UpdateImageSetMetadata solicitudes simultáneas por almacén de datos	Cada región admitida: 100	Sí	El número máximo de UpdateImageSetMetadata solicitudes simultáneas por almacén de datos en la región actual AWS
Número máximo de trabajos de importación simultáneos por almacén de datos	ap-southeast-2: 20	Sí	El número máximo de trabajos de importación simultáneos por banco de

Nombre	Valor predeterminado	Ajusta	Descripción
	Cada una de las demás regiones compatibles: 100		datos en la región actual AWS
Máximo de almacenes de datos	Cada región admitida: 10	Sí	El número máximo de almacenes de datos activos en la región actual AWS
Número máximo de ImageFrames copias que se pueden copiar por CopyImageSet solicitud	Cada región admitida: 1000	Sí	El número máximo de ImageFrames copias que se pueden copiar por CopyImageSet solicitud en la AWS región actual
Número máximo de archivos en un trabajo DICOM de importación	Cada región admitida: 5000	Sí	El número máximo de archivos en un trabajo de DICOM importación en la AWS región actual
Número máximo de carpetas anidadas en un trabajo de DICOM importación	Cada región admitida: 10 000	No	El número máximo de carpetas anidadas en un trabajo de DICOM importación en la región actual AWS
El límite máximo de tamaño de carga útil (en KB) aceptado por UpdateImageSetMetadata	Cada región admitida: 10 KB	Sí	El límite máximo de tamaño de carga útil (en KB) aceptado UpdateImageSetMetadata en la región actual AWS

Nombre	Valor predeterminado	Ajusta	Descripción
Tamaño máximo (en GB) de todos los archivos de un trabajo de DICOM importación	Cada región admitida: 10 gigabytes	No	El tamaño máximo (en GB) de todos los archivos de un trabajo de DICOM importación en la AWS región actual
Tamaño máximo (en GB) de cada archivo DICOM P10 de un trabajo de DICOM importación	Cada región admitida: 4 gigabytes	No	El tamaño máximo (en GB) de cada archivo DICOM P10 del trabajo de DICOM importación en la región actual AWS
Límite de tamaño máximo (en MB) ImageSetMetadata por importación, copia y UpdateImageSet	Cada región admitida: 50 MB	Sí	El límite de tamaño máximo (en MB) ImageSetMetadata por importación, copia y UpdateImageSet en la AWS región actual

AWS HealthImaging límites de estrangulamiento

Tu AWS cuenta tiene límites de limitación que se aplican a las acciones. AWS HealthImaging API En todas las acciones se genera un error `ThrottlingException` si se superan los límites de limitación. Para obtener más información, consulta la AWS HealthImaging API Referencia.

> **Note**
>
> Los límites de regulación se pueden ajustar para todas las HealthImaging API acciones. Para solicitar un ajuste del límite de limitación, póngase en contacto con el Centro de soporte de AWS.

En la siguiente tabla se enumeran los límites de limitación para AWS HealthImaging API las acciones y representaciones de servicios nativas. DICOMweb

AWS HealthImaging límites de estrangulamiento

Acción	Velocidad de limitación	Ráfaga del limitación
CreateDatastore	0,085 TPS	1 TPS
GetDatastore	10 TPS	20 TPS
ListDatastores	5 TPS	10 TPS
DeleteDatastore	0,085 TPS	1 TPS
Un tartDICOMImport trabajo	0,25 TPS	1 TPS
G etDICOMImport Job	25 TPS	50 TPS
istDICOMImportEmpleos L	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
G etDICOMInstance *	50 TPS	100 TPS

Acción	Velocidad de limitación	Ráfaga del limitación
etDICOMInstanceMetadatos G*	50 TPS	100 TPS
Marcos G* etDICOMInstance	50 TPS	100 TPS

*Representación de un servicio DICOMweb

AWS HealthImaging proyectos de muestra

AWS HealthImaging proporciona los siguientes proyectos de muestra sobre GitHub.

DICOMIngestión desde las instalaciones hasta AWS HealthImaging

Un proyecto AWS sin servidor para implementar una solución perimetral de IoT que reciba DICOM archivos de una DICOM DIMSE fuente (PACS,VNA, escáner CT) y los almacene en un bucket seguro de Amazon S3. La solución indexa los DICOM archivos de una base de datos y pone en cola cada DICOM serie para importarla. AWS HealthImaging Se compone de un componente que se ejecuta en la periferia y es gestionado por AWS IoT Greengrassuna canalización de DICOM ingestión que se ejecuta en la nube. AWS

Tile Level Marker (TLM) Proxy

Un AWS Cloud Development Kit (AWS CDK)proyecto para recuperar fotogramas de imágenes AWS HealthImaging mediante marcadores de nivel de mosaico (TLM), una función de High-Throughput JPEG 2000 (). HTJ2K Esto se traduce en tiempos de recuperación más rápidos con imágenes de menor resolución. Los posibles flujos de trabajo incluyen la generación de miniaturas y la carga progresiva de imágenes.

Amazon CloudFront Delivery

Un proyecto AWS sin servidor para crear una CloudFront distribución de Amazon con un HTTPS punto final que almacene en caché (mediante el usoGET) y entregue marcos de imágenes desde el borde. De forma predeterminada, el punto de conexión autentica las solicitudes con un token JWT web de Amazon JSON Cognito (). Tanto la autenticación como la firma de solicitudes se realizan en la periferia mediante Lambda@Edge. Este servicio es una función de Amazon CloudFront que permite ejecutar el código más cerca de los usuarios de la aplicación, lo que mejora el rendimiento y reduce la latencia. No hay ninguna infraestructura que gestionar.

AWS HealthImaging Interfaz de usuario del visor

Un AWS Amplifyproyecto para implementar una interfaz de usuario front-end con autenticación de backend con la que se pueden ver los atributos de los metadatos del conjunto de imágenes y los marcos de imágenes (datos de píxeles) almacenados AWS HealthImaging mediante una decodificación progresiva. Si lo desea, puede integrar los proyectos Tile Level Marker (TLM) Proxy o Amazon CloudFront Delivery anteriores para cargar marcos de imágenes mediante un método alternativo.

AWS HealthImaging DICOMwebProxy

Un proyecto basado en Python para habilitar los puntos finales DICOMweb WADO -RS y QIDO -RS en un almacén de HealthImaging datos para admitir visores de imágenes médicas basados en la web y otras aplicaciones compatibles. DICOMweb

> **Note**
>
> Este proyecto no utiliza la representación descrita en. HealthImaging DICOMweb APIs Uso DICOMweb con AWS HealthImaging

Para ver otros proyectos de muestra, consulte los AWS HealthImaging ejemplos en GitHub.

HealthImaging Utilización con un AWS SDK

AWS Los kits de desarrollo de software (SDK) están disponibles para muchos lenguajes de programación populares. Cada SDK proporciona una API, ejemplos de código y documentación que facilitan a los desarrolladores la creación de aplicaciones en su lenguaje preferido.

Documentación de SDK	Ejemplos de código
AWS SDK for C++	AWS SDK for C++ ejemplos de código
AWS CLI	AWS CLI ejemplos de código
AWS SDK for Go	AWS SDK for Go ejemplos de código
AWS SDK for Java	AWS SDK for Java ejemplos de código
AWS SDK for JavaScript	AWS SDK for JavaScript ejemplos de código

Documentación de SDK	Ejemplos de código
AWS SDK para Kotlin	AWS SDK para Kotlin ejemplos de código
AWS SDK for .NET	AWS SDK for .NET ejemplos de código
AWS SDK for PHP	AWS SDK for PHP ejemplos de código
AWS Tools for PowerShell	Herramientas para ejemplos PowerShell de código
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) ejemplos de código
AWS SDK for Ruby	AWS SDK for Ruby ejemplos de código
AWS SDK para Rust	AWS SDK para Rust ejemplos de código
AWS SDK para SAP ABAP	AWS SDK para SAP ABAP ejemplos de código
AWS SDK para Swift	AWS SDK para Swift ejemplos de código

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicite un ejemplo de código a través del enlace de Enviar comentarios que se encuentra al final de esta página.

AWS HealthImaging lanzamientos

En la siguiente tabla se muestra cuándo se publicaron las funciones y actualizaciones del AWS HealthImaging servicio y de la documentación. Para más información sobre una versión, consulte el tema enlazado.

Cambio	Descripción	Fecha
Volver a un ID de versión anterior del conjunto de imágenes	HealthImaging proporciona el `revertToVersionId` parámetro para volver a un ID de versión anterior del conjunto de imágenes. Para obtener más información, consulte `revertToVersionId` la AWS HealthImaging APIReferencia.	24 de julio de 2024
Función de fuerza para la modificación del conjunto de imágenes	HealthImaging proporciona el tipo de `Overrides` datos con el parámetro de `forced` solicitud opcional. La configuración de este parámetro fuerza las `CopyImageSet` acciones `UpdateImageSetMetadata` `data` y, incluso si los metadatos a nivel de paciente, estudio o serie no coinciden . Para obtener más información, consulte Modificaciones en la AWS HealthImaging API referencia. • `UpdateImageSetMetadata` `data` funcionalidad de fuerza: HealthImaging introduce el parámetro de	24 de julio de 2024

`force` solicitud opcional para actualizar los siguiente s atributos:

- `Tag.StudyInstanceU ID` , `Tag.Serie sInstanceUID` , `Tag.SOPInstanceUID` , y `Tag.StudyID`
- Añadir, eliminar o actualizar elementos de DICOM datos privados a nivel de instancia

Para obtener más informaci ón, consulte UpdateIma geSetMetadatala AWS HealthImaging APIRefere ncia.

- `CopyImageSet` funcional idad de fuerza: HealthIma ging introduce el parámetro de `force` solicitud opcional para copiar conjuntos de imágenes. Si se establece este parámetro, se fuerza la `CopyImageSet` acción, incluso si los metadatos a nivel de paciente, estudio o serie no coinciden entre sí`sourceImageSet` . `destinationImageSe t` En estos casos, los metadatos incoherentes permanecen inalterados en el`destinationImageSe t` . Para obtener más

	información, consulte CopyImageSetla AWS HealthImaging APIReferencia.	
Copie subconjuntos de instancias SOP	HealthImaging mejora la `CopyImageSet` acción para que pueda elegir una o más SOP instancias de una `sourceImageSet` para copiarlas a una`destinationImageSet` . Para obtener más información, consulte Copia de conjuntos de imágenes.	24 de julio de 2024
`GetDICOMInstanceMetadata` para devolver los metadatos de la DICOM instancia	HealthImaging proporciona el `GetDICOMInstanceMetadata` servicio de devolución de los metadatos de la DICOM parte 10 (`.json`archivo). Para obtener más información, consulte Obtener metadatos de DICOM instancias de HealthImaging.	11 de julio de 2024
`GetDICOMInstanceFrames` para devolver fotogramas de DICOM instancia (datos de píxeles)	HealthImaging proporciona el `GetDICOMInstanceFrames` servicio de devolución de los fotogramas de la DICOM Parte 10 (`multipart` solicitud). Para obtener más información, consulte Obtener marcos de DICOM instancia de HealthImaging.	11 de julio de 2024

Soporte mejorado para la importación de DICOM datos no estándar	HealthImaging proporciona soporte para la importación de datos que incluyen desviaciones del DICOM estándar. Para obtener más información, consulte DICOMrestricciones de elementos.
	• Los siguientes elementos de DICOM datos pueden tener una longitud máxima de 256 caracteres:
	  • `Patient's Name (0010,0010)`
	  • `Patient ID (0010,0020)`
	  • `Accession Number (0008,0050)`
	• Se permiten las siguientes variaciones de sintaxis para `Study Instance UID` `Series Instance UIDTreatment Session UID`,`Manufacturer’s Device Class UID`,`Device UID`, y`Acquisition UID` :
	  • El primer elemento de cualquiera UID puede ser cero
	  • UIDspuede empezar con uno o más ceros a la izquierda
		28 de junio de 2024

	• UIDspuede tener una longitud máxima de 256 caracteres	
Notificaciones de eventos	HealthImaging se integra con Amazon EventBridge para dar soporte a aplicaciones basadas en eventos. Para obtener más información, consulte Uso de Amazon EventBridge con HealthImaging.	5 de junio de 2024
`GetDICOMInstance` para devolver DICOM datos de instancias	HealthImaging proporciona el `GetDICOMInstance` servicio para devolver los datos de la instancia de la DICOM Parte 10 (`.dcm`archivo). Para obtener más información, consulte Obtener una DICOM instancia de HealthImaging.	15 de mayo de 2024
Importación multicuenta	HealthImaging proporciona soporte para la importación de datos desde buckets de Amazon S3 ubicados en otras regiones compatibles. Para obtener más información, consulte Importación multicuenta para AWS HealthImaging.	15 de mayo de 2024

Mejoras de búsqueda para conjuntos de imágenes	HealthImaging `SearchImageSets` La acción admite las siguientes mejoras de búsqueda. Para obtener más información, consulte Búsqueda de conjuntos de imágenes. • Soporte adicional para buscar en `UpdatedAt` y `SeriesInstanceUID` • Busque entre la hora de inicio y la hora de finalización • Ordena los resultados de la búsqueda por `Ascending` o `Descending` • DICOMLos parámetros de la serie se devuelven en las respuestas	3 de abril de 2024
Se ha aumentado el tamaño máximo de archivo para las importaciones	HealthImaging admite un tamaño de archivo máximo de 4 GB para cada archivo DICOM P10 de un trabajo de importación. Para obtener más información, consulte Service Quotas.	6 de marzo de 2024

Transfiera las sintaxis para JPEG Lossless y HTJ2K	HealthImaging proporciona soporte para las siguientes sintaxis de transferencia para la importación de trabajos. Para obtener más información, consulte Sintaxis de transferencia compatibles. • 1.2.840.10008.1.2.4.57 — Sin pérdidas y no jerárquico (proceso 14) JPEG • 1.2.840.10008.1.2.4.201: compresión de imágenes 2000 de alto JPEG rendimiento (solo sin pérdidas) • 1.2.840.10008.1.2.4.202: RPCL compresión de imágenes de JPEG 2000 de alto rendimiento con opciones (solo sin pérdidas) • 1.2.840.10008.1.2.4.203: compresión de imágenes de JPEG 2000 de alto rendimiento	16 de febrero de 2024
Ejemplos de código probados	HealthImaging la documentación proporciona ejemplos de código probados AWS SDKs para AWS CLI y para Python JavaScript, Java y C++. Para obtener más información, consulte Ejemplos de código para HealthImaging usar AWS SDKs.	19 de diciembre de 2023

Se ha aumentado el número máximo de archivos para las importaciones	HealthImaging admite hasta 5000 archivos para un solo trabajo de importación. Para obtener más información, consulte Service Quotas.	19 de diciembre de 2023
Carpetas anidadas para importaciones	HealthImaging admite hasta 10 000 carpetas anidadas para un solo trabajo de importación. Para obtener más información, consulte Service Quotas.	1 de diciembre de 2023
Importaciones más rápidas	HealthImaging permite importar 20 veces más rápido en todas las regiones compatibles. Para obtener más información, consulte Puntos de conexión de servicio.	1 de diciembre de 2023
CloudFormation soporte	HealthImaging admite la infraestructura como código (IaC) para el aprovisio namiento de almacenes de datos. Para obtener más información, consulte Creación de recursos de AWS HealthImaging con AWS CloudFormation.	21 de septiembre de 2023

Disponibilidad general	AWS HealthImaging está disponible para todos los clientes de las regiones EE.UU. Este (Norte de Virginia), EE.UU. Oeste (Oregón), Europa (Irlanda) y Asia Pacífico (Sídney).	26 de julio de 2023

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.