



Guía para desarrolladores

AWS HealthImaging



AWS HealthImaging: Guía para desarrolladores

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

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

¿Qué es AWS HealthImaging?	1
Aviso importante	3
Características	3
Servicios relacionados	5
Acceso	5
HIPAA	6
Precios	6
Introducción	7
Conceptos	7
Almacén de datos	7
Conjuntos de imágenes	8
Metadatos	8
Marcos de imágenes	8
Configuración	9
Inscríbese en una Cuenta de AWS	9
Creación de un usuario con acceso administrativo	10
Creación de buckets de S3	11
Crear un almacén de datos	12
Creación de un usuario de IAM	12
Creación de un rol de IAM	13
Instale el AWS CLI	15
Tutorial	16
Administración de almacenes de datos	18
Creación de un almacén de datos	18
Obtención de propiedades de los almacenes de datos	27
Enumeración de almacenes de datos	35
Eliminación de almacenes de datos	43
Usando DICOMweb	51
Almacenamiento de instancias con STOW-RS	51
Recuperación de datos con WADO-RS	55
Recupera una instancia	56
Recuperar metadatos de instancia	59
Recupera los metadatos de la serie	60
Recuperar marcos	62

Recupere datos masivos	64
Búsqueda de datos con QIDO-RS	66
DICOMweb APIs buscar HealthImaging	67
Tipos de DICOMweb consulta compatibles para HealthImaging	68
Busque estudios	71
Busque series	73
Busque instancias	74
Autenticación OIDC	76
¿Cómo funciona la verificación de tokens	76
Requisitos y configuración	80
Importación de datos de imágenes	91
Introducción a los trabajos de importación	91
Inicio de un trabajo de importación	95
Obtención de las propiedades de trabajos de importación	104
Enumeración de trabajos de importación	111
Acceso a conjuntos de imágenes	118
Descripción de los conjuntos de imágenes	118
Búsqueda de conjuntos de imágenes	126
Obtención de las propiedades del conjunto de imágenes	152
Obtención de metadatos de conjuntos de imágenes	159
Obtención de datos de píxeles de los conjuntos de imágenes	170
Modificación de conjuntos de imágenes	179
Listado de versiones de conjuntos de imágenes	179
Actualización de los metadatos de un conjunto de imágenes	187
Para actualizar los metadatos de un conjunto de imágenes principal	206
Para hacer que un conjunto de imágenes no principal sea principal	208
Copia de conjuntos de imágenes	208
Eliminación de un conjunto de imágenes	224
Etiquetado de recursos	232
Etiquetado de un recurso	232
Listado de etiquetas de un recurso	238
Eliminación de las etiquetas de un recurso	243
Ejemplos de código	250
Conceptos básicos	251
Hola HealthImaging	252
Acciones	257

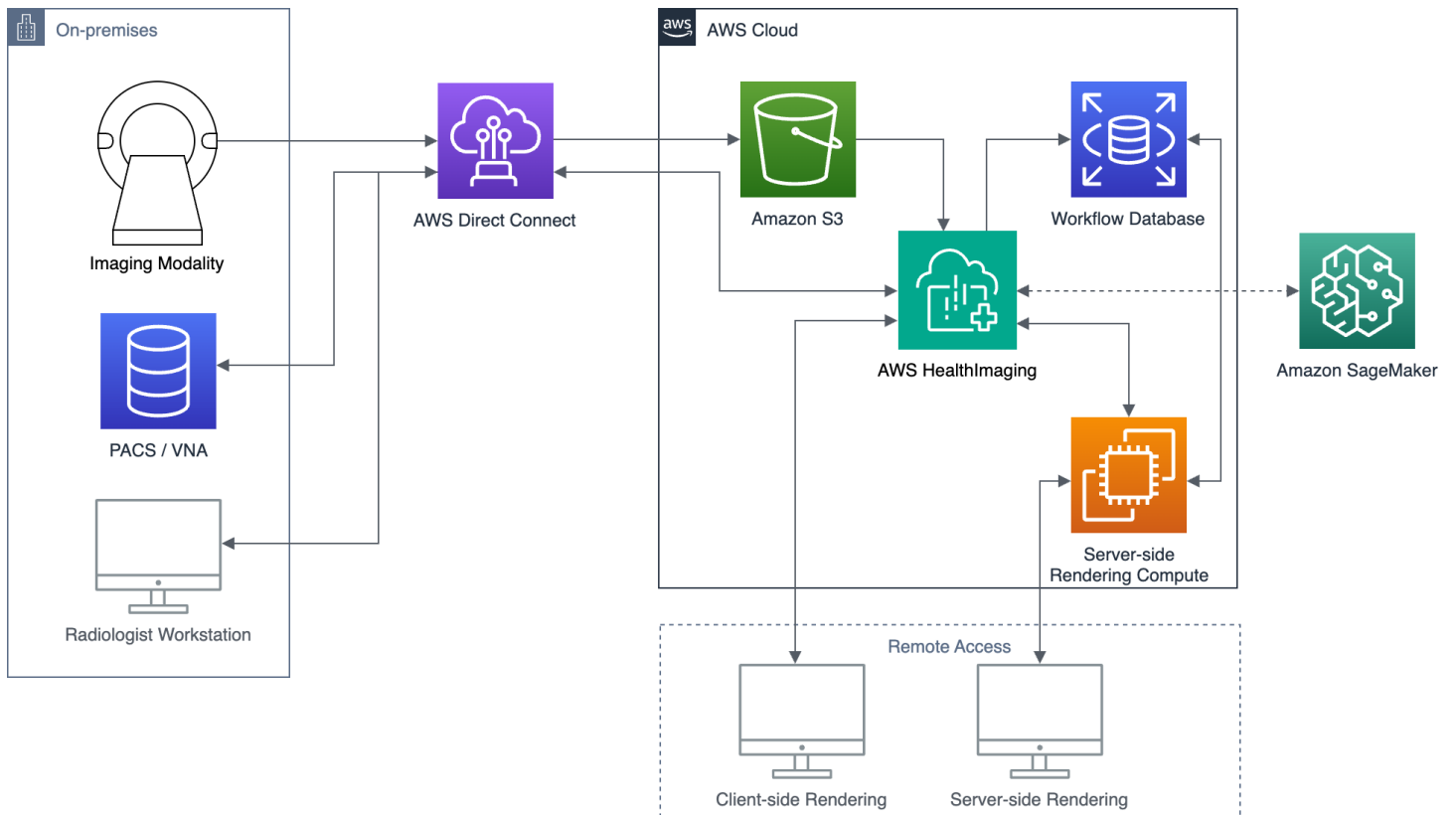
Escenarios	413
Introducción a los conjuntos y marcos de imágenes	413
Etiquetar un almacén de datos	468
Etiquetar un conjunto de imágenes	478
Supervisión	489
CloudTrail (Llamadas a la API)	490
Creating a trail	490
Descripción de las entradas de los entradas de registro	492
CloudWatch (Métricas)	493
Visualización de HealthImaging métricas	494
Crear una alarma de	494
EventBridge (Eventos)	495
HealthImaging eventos enviados a EventBridge	495
HealthImaging estructura y ejemplos de eventos	497
Seguridad	514
Protección de datos	515
Cifrado de datos	516
Privacidad del tráfico de red	526
Gestión de identidad y acceso	526
Público	527
Autenticación con identidades	527
Administración del acceso con políticas	529
Cómo HealthImaging funciona AWS con IAM	530
Ejemplos de políticas basadas en identidades	537
AWS políticas gestionadas	540
Prevención de la sustitución confusa entre servicios	542
Resolución de problemas	543
Validación de conformidad	545
Seguridad de la infraestructura	546
Infraestructura como código	547
HealthImaging y CloudFormation plantillas	547
Obtenga más información sobre CloudFormation	547
Puntos de conexión de VPC	548
Consideraciones para los puntos de conexión de VPC	548
Creación de un punto de conexión de VPC	548
Creación de una política de punto de conexión de VPC	549

Importación multicuenta	550
Resiliencia	552
Referencia	553
DICOM	553
Clases de SOP compatibles	554
Normalización de metadatos	554
Sintaxis de transferencia compatibles	559
Restricciones de los elementos DICOM	562
Restricciones de los metadatos de DICOM	563
HealthImaging	564
Cuotas y puntos de conexión	564
Límites de limitación	570
Verificación de datos de píxeles	572
Códigos de advertencia	574
Bibliotecas de decodificación de marcos de imágenes	578
Proyectos de ejemplo	580
Trabajando con AWS SDKs	581
Optimización de costes	583
Cómo funciona la organización inteligente en niveles	583
Estimación del almacenamiento de datos estructurados	584
Versiones	586
.....	dcii

¿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 escala de petabytes. HealthImaging los 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 incorporada. Nube de AWS

Temas

- [Aviso importante](#)
- [Características de AWS HealthImaging](#)
- [Servicios relacionados AWS](#)
- [Acceso a AWS HealthImaging](#)
- [Cumplimiento con para la HIPAA y seguridad de la información](#)
- [Precios](#)

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 capacitados 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 individuales de los metadatos utilizando palabras clave fáciles de usar en lugar de números hexadecimales desconocidos. group/element 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 como imágenes JPEG 2000 (HTJ2K) de alto rendimiento, un códec de compresión de imágenes avanzado. HTJ2K aprovecha las ventajas de una sola instrucción y varios datos (SIMD) de los procesadores modernos para ofrecer nuevos niveles de rendimiento. HTJ2K es un orden de magnitud más rápido que JPEG2 000 y al menos dos veces más 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. Para obtener más información, consulte [Sintaxis de transferencia compatibles](#).

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 [DICOM](#).

Jerarquía de datos DICOM gestionada por el servicio

AWS organiza HealthImaging automáticamente los datos DICOM P10 al importarlos por elementos de datos DICOM a nivel de paciente, estudio y serie. El servicio organiza estos datos DICOM en conjuntos de imágenes que corresponden a las series DICOM, lo que simplifica los flujos de trabajo posteriores a la importación. La organización a nivel de estudio y serie se mantiene a medida que se importan nuevos datos.

DICOMweb Compatibilidad de API

AWS HealthImaging ofrece DICOMweb compatibilidad APIs para simplificar las integraciones y permitir la interoperabilidad con las aplicaciones existentes. El servicio también ofrece opciones nativas de la nube APIs que permiten realizar acciones no compatibles con el DICOMweb estándar, como las operaciones de actualización de metadatos.

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 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 Consola de administración de AWS, AWS Command Line Interface y AWS SDKs. Esta guía proporciona instrucciones de procedimiento Consola de administración de AWS y ejemplos de código para AWS CLI y AWS SDKs.

Consola de administración de AWS

Consola de administración de AWS Proporciona una interfaz de usuario basada en la web para administrar 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 SDKs

AWS SDKs proporciona 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 sobre las que construir AWS](#).

Solicitudes HTTP

Puede HealthImaging realizar acciones mediante solicitudes HTTP, pero debe especificar distintos puntos de enlace en función del tipo de acciones que se utilicen. Para obtener más información, consulte [Acciones de la API admitidas para 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 conformidad para 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 [Optimización de costes](#).

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 usar el [AWS CLI](#) o el [AWS SDKs](#), debe instalarlos y configurarlos.

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

Temas

- [HealthImaging Conceptos de AWS](#)
- [Configuración de AWS HealthImaging](#)
- [HealthImaging Tutorial de AWS](#)

HealthImaging Conceptos de AWS

La terminología y los conceptos siguientes son fundamentales para comprender 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. Una AWS cuenta puede tener cero o varios almacenes de datos. Cada almacén de datos tiene su propia clave de cifrado de 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 [Optimización de costes](#).

Conjuntos de imágenes

Un conjunto de imágenes es un AWS concepto que define un mecanismo de agrupamiento abstracto para optimizar los datos relacionados con las imágenes médicas. 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). HealthImaging intenta organizar los datos importados de acuerdo con la jerarquía DICOM de estudio, serie e instancia. [Las instancias DICOM que se añaden correctamente a la jerarquía HealthImaging gestionada se denominan conjuntos de imágenes principales. La importación de datos de DICOM P10 permite: crear un nuevo conjunto de imágenes principal; fusionar las instancias en un conjunto de imágenes principal existente si las instancias ya existen en la colección principal; o, en caso de conflictos entre elementos de metadatos, crear un nuevo conjunto de imágenes no principal.](#)

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 a él rápidamente. Esto resulta útil para los visualizadores de imágenes, los análisis y los casos de AI/ML uso que no requieren datos de 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. Algunos archivos conservan su codificación de sintaxis de transferencia original durante la importación, mientras que otros se transcodifican. Los almacenes de datos de Amazon Web Services se pueden configurar para transcodificar fotogramas de imágenes sin pérdidas a JPEG 2000 (HTJ2K) sin pérdidas de alto rendimiento o JPEG 2000 sin pérdidas. Si un marco de imagen está codificado en HTJ2 K o JPEG 2000, debe decodificarse

antes de visualizarlo en un visor de imágenes. Para obtener más información, consulte [Sintaxis de transferencia compatibles](#), [Obtención de datos de píxeles de los conjuntos de imágenes](#) y [Bibliotecas de decodificación de marcos de imágenes](#).

Configuración de AWS HealthImaging

Debe configurar su AWS entorno antes de utilizar AWS HealthImaging. Antes de completar el [tutorial](#) de la sección siguiente, debe haber los temas que se detallan a continuación.

Temas

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

Inscríbase en una Cuenta de AWS

Si no tiene una Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirse 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 o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

Cuando te registras en una Cuenta de AWS, Usuario raíz de la cuenta de AWS se crea un. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa 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 crear un usuario administrativo Cuenta de AWS, asegúrelo Usuario raíz de la cuenta de AWS AWS IAM Identity Center, habilite y cree un usuario administrativo para no usar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión [Consola de administración de AWS](#) como propietario de la cuenta seleccionando el usuario root e introduciendo su dirección de Cuenta de AWS correo electrónico. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In .

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitar un dispositivo MFA virtual para el usuario Cuenta de AWS raíz \(consola\)](#) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center .

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre su uso 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](#) Guía del AWS IAM Identity Center usuario.

Inicio de sesión como usuario con acceso de administrador

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

Para obtener ayuda para iniciar sesión con un usuario del Centro de identidades de IAM, consulte [Iniciar sesión en el portal de AWS acceso](#) en la Guía del AWS Sign-In 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 de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center .

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

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center .

Creación de buckets de S3

Para importar datos de DICOM P10 a 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 la política AWS Identity and Access Management (IAM), los nombres de los 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 el [rol de IAM para importar](#).

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

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 HealthImaging datos](#) de AWS 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 usuario de IAM con HealthImaging permiso de acceso total

Práctica recomendada

Le sugerimos que cree usuarios de IAM independientes para diferentes necesidades, como la importación, el acceso a los datos y la administración de los mismos, siguiendo las recomendaciones de [Conceder el acceso con el privilegio mínimo](#) del Marco de AWS Well-Architected.

Para los fines del [tutorial](#) de la sección siguiente, se utilizará un único usuario de IAM.

Para crear un usuario de IAM

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

 Note

Los permisos de IAM se pueden restringir. Para obtener más información, consulte [AWS políticas administradas para AWS HealthImaging](#).

Creación un rol de IAM para la importación

 Note

Las siguientes instrucciones hacen referencia a un rol AWS Identity and Access Management (IAM) que concede acceso de lectura y escritura a los buckets de Amazon S3 para importar sus datos DICOM. Si bien el rol es obligatorio para el [tutorial](#) de la sección siguiente, le recomendamos que añada permisos de IAM a los usuarios, grupos y roles que utilicen [AWS políticas administradas para AWS HealthImaging](#). Le resultará más fácil usarlos que escribir las políticas usted mismo.

Un rol de IAM es una identidad de IAM que puede crear en su cuenta y que tiene permisos específicos. Para iniciar un trabajo de importación, el rol de IAM que llama a la acción `StartDICOMImportJob` debe estar asociado a una política de usuario que conceda acceso a los buckets de Amazon S3 que se utilizan para leer los 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 permita HealthImaging a AWS asumir la función.

Cómo crear un rol de IAM con fines de importación

1. Use la [consola de IAM](#) para crear un rol denominado `ImportJobDataAccessRole`. Este rol se utiliza para el [tutorial](#) de la siguiente sección. Para más información, consulte [Creación de roles de IAM](#) en la Guía del usuario de IAM.

 Tip

A los efectos de esta guía, los códigos de ejemplo de [Inicio de un trabajo de importación](#) hacen referencia al rol de IAM `ImportJobDataAccessRole`.

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

JSON

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

3. Adjunte la siguiente relación de confianza (política) al rol de IAM `ImportJobDataAccessRole`. 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 HealthImaging datos de AWS, pero con buckets de Amazon S3 específicos del almacén de datos, funciones de IAM y políticas de confianza.

 Note

El `Condition` bloqueo de esta política de confianza ayuda a evitar el confuso problema de los diputados al garantizar que solo se pueda acceder a su almacén de HealthImaging datos específico de AWS. Para obtener más información sobre esta medida de seguridad, consulte [Prevención de problemas de tráfico entre servicios en HealthImaging](#).

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

Para más información acerca de los roles de IAM, consulte [Roles de IAM](#) en la Guía del usuario de IAM. Para más información acerca de las políticas y permisos de IAM, consulte [Permisos y políticas](#) en la Guía del usuario de IAM.

Instale el AWS CLI (opcional)

Si utiliza AWS Command Line Interface, se requiere el siguiente procedimiento. Si utiliza el Consola de administración de AWS o AWS SDKs, puede omitir el siguiente procedimiento.

Para configurar el AWS CLI

1. Descargue y configure la AWS CLI. Para obtener instrucciones, consulte los siguientes temas en la Guía del usuario de AWS Command Line Interface .
 - [Instalar o actualizar la última versión del AWS CLI](#)
 - [Cómo empezar con el AWS CLI](#)
2. En el AWS CLI `config` archivo, añada un perfil con nombre para el administrador. Este perfil se utiliza al ejecutar los AWS CLI comandos. Según el principio de seguridad de privilegio mínimo, le recomendamos que cree un rol de IAM independiente con privilegios específicos para las tareas que se estén realizando. Para más información sobre los perfiles con nombre,

consulte [Opciones de los archivos de configuración y credenciales](#) en la Guía del usuario de AWS Command Line Interface .

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

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

```
aws medical-imaging help
```

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

HealthImaging Tutorial de AWS

Objetivo

El objetivo de este tutorial es importar binarios (`.dcm`archivos) de DICOM P10 a un [almacén de HealthImaging datos](#) de AWS y transformarlos en [conjuntos de imágenes](#) compuestos por [metadatos](#) y [marcos de imágenes](#) (datos de píxeles). [Tras importar los datos DICOM, utiliza 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](#)

Administración de almacenes de datos con AWS HealthImaging

Con AWS HealthImaging, 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 Consola de administración de AWS, la CLI de AWS, y los SDKs de AWS.

Note

El último tema de este capítulo trata sobre la [optimización de costes](#). Tras 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](#)
- [Obtención de propiedades de los almacenes de datos](#)
- [Enumeración de almacenes de datos](#)
- [Eliminación de almacenes de datos](#)

Creación de un almacén de datos

Utilice la `CreateDatastore` acción para crear un [almacén de HealthImaging datos de AWS](#) para importar archivos DICOM P10. Los siguientes menús proporcionan un procedimiento Consola de administración de AWS y ejemplos de código para y. `AWS CLI` `AWS SDKs` Para obtener más información, consulte [CreateDatastore](#) la referencia de la HealthImaging API de AWS. Al crear un banco de datos, puede seleccionar la sintaxis de transferencia predeterminada que AWS HealthImaging utilizó para transcodificar y almacenar marcos de imágenes sin pérdidas. Esta configuración no se puede cambiar una vez creado el banco de datos.

JPEG 2000 (HTJ2K) de alto rendimiento

HTJ2K (JPEG 2000 de alto rendimiento) es el formato de almacenamiento predeterminado para HealthImaging los almacenes de datos. Es una extensión del estándar JPEG 2000 que ofrece un rendimiento de codificación y decodificación significativamente mejorado. Al crear un almacén de datos sin especificar un `–lossless-storage-format`, utiliza HealthImaging automáticamente K. HTJ2 Consulte la CLI de AWS y SDKs la sección siguiente para crear un almacén de datos mediante HTJ2 K.

JPEG 2000 sin pérdidas

La codificación JPEG 2000 Lossless permite crear almacenes de datos que conservan y recuperan fotogramas de imágenes sin pérdidas en formato JPEG 2000 sin necesidad de transcódificarlos, lo que permite una recuperación con menor latencia en las aplicaciones que requieren JPEG 2000 Lossless (DICOM Transfer Syntax UID 1.2.840.10008.1.2.4.90), consulte para obtener más información. [Sintaxis de transferencia compatibles](#) Consulte la CLI de AWS y SDKs la sección siguiente para crear un banco de datos con el formato JPEG 2000 sin pérdidas.

Importante

- No nombre los almacenes de datos utilizando información médica protegida (PHI), información de identificación personal (PII) ni ningún otro tipo de información confidencial o sensible.
- La consola de AWS admite la creación de almacenes de datos con la configuración predeterminada. Use la AWS CLI o el AWS SDK para crear un banco de datos con una opción `–lossless-storage-format` especificada.

Cómo crear un almacén de datos

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

AWS Consola

1. Abra la [página de creación de 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 sus 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 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 getopt command in a function.

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

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

# Retrieve the calling parameters.
while 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 obtener detalles sobre la API, consulte [CreateDatastore](#) la Referencia de AWS CLI comandos.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

CLI

AWS CLI

Ejemplo 1: 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`. Al crear un almacén de datos sin especificar `--lossless-storage-format`, el AWS HealthImaging valor predeterminado es HTJ2 K (JPEG 2000 de alto rendimiento).

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

Salida:

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

Ejemplo 2: Creación de un almacén de datos con el formato de almacenamiento JPEG 2000 sin pérdidas

Un almacén de datos configurado con el formato de almacenamiento JPEG 2000 sin pérdidas transcodificará y conservará los fotogramas de imagen sin pérdidas en formato JPEG 2000. A continuación, los fotogramas de imagen se pueden recuperar en JPEG 2000 sin pérdidas sin necesidad de transcodificación. En el ejemplo de código `create-datastore` siguiente,

se crea un almacén de datos configurado para el formato de almacenamiento JPEG 2000 sin pérdidas con el nombre my-datastore.

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

Salida:

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

Para obtener más información, consulte [Creación de un banco de datos en la Guía para desarrolladores](#).AWS HealthImaging

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

```
}
```

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [CreateDatastore](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

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

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Obtención de propiedades de los almacenes de datos

Utilice la GetDatastore acción para recuperar las propiedades del [almacén HealthImaging de datos](#) de AWS. Los siguientes menús proporcionan un procedimiento Consola de administración de AWS y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte [GetDatastore](#) la [referencia de la HealthImaging API de AWS](#).

Cómo obtener propiedades de los almacenes de datos

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

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

```

```

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

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

local response

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

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

echo "$response"

return 0
}

```

- Para obtener detalles sobre la API, consulte [GetDatastore](#) la Referencia de AWS CLI comandos.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

CLI

AWS CLI

Ejemplo 1: 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",  
    "losslessStorageFormat": "HTJ2K"  
    "datastoreArn": "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012",  
    "createdAt": "2022-11-15T23:33:09.643000+00:00",  
    "updatedAt": "2022-11-15T23:33:09.643000+00:00"  
  }  
}
```

Ejemplo 2: Para configurar las propiedades del almacén de datos para JPEG2 000

En el ejemplo de código `get-datastore` siguiente, se obtienen las propiedades de un almacén de datos configurado para el formato de almacenamiento JPEG 2000 sin pérdidas.

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

Salida:

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

Para obtener más información, consulte [Obtener las propiedades del almacén de datos](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetDatastore](#) la Referencia de AWS CLI comandos.

Java**SDK para Java 2.x**

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

    return null;
}
```

- Para obtener más información sobre la API, consulte [GetDatastore](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

```
//      updatedAt: 2023-08-04T18:50:36.239Z
//    }
//  }
return response.datastoreProperties;
};
```

- Para obtener más información sobre la API, consulte [GetDatastore](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

```

    )
    raise
else:
    return data_store["datastoreProperties"]

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [GetDatastore](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->getdatastore( iv_datastoreid = iv_datastore_id ).
    DATA(lo_properties) = oo_result->get_datastoreproperties( ).
    DATA(lv_name) = lo_properties->get_datastorename( ).
    DATA(lv_status) = lo_properties->get_datastorestatus( ).
    MESSAGE 'Data store properties retrieved.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Data store not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.

```

```
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [GetDatastore](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Enumeración de almacenes de datos

Utilice la `ListDatastores` acción para enumerar [los almacenes de datos](#) disponibles en AWS HealthImaging. Los siguientes menús proporcionan un procedimiento Consola de administración de AWS y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte [ListDatastores](#) la referencia de la HealthImaging API de AWS.

Cómo enumerar almacenes de datos

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

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

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

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

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

echo "$response"

return 0
}
```

- Para obtener detalles sobre la API, consulte [ListDatastores](#) la Referencia de AWS CLI comandos.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta la sección sobre la [lista de almacenes de datos](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [ListDatastores](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

```

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

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

    return null;
}

```

- Para obtener más información sobre la API, consulte [ListDatastores](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

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

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

    /**
     * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
     */
    const datastoreSummaries = [];

```


Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [ListDatastores](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

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

- Para obtener más información sobre la API, consulte [ListDatastores](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Eliminación de almacenes de datos

Utilice la `DeleteDatastore` acción para eliminar un [almacén de HealthImaging datos de AWS](#). Los siguientes menús proporcionan un procedimiento Consola de administración de AWS y ejemplos de código para el AWS CLI y AWS SDKs. Para obtener más información, consulte [DeleteDatastore](#) la referencia de la HealthImaging API de AWS.

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 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.
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 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 getopt command in a function.

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

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

```
export OPTIND=1

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

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

local error_code=${?}

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

return 0
}
```

- Para obtener detalles sobre la API, consulte [DeleteDatastore](#) la Referencia de AWS CLI comandos.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta [Eliminar un banco de datos](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

  return response;
};
```

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [DeleteDatastore](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

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

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace [Enviar comentarios](#) en la barra lateral derecha de esta página.

Uso DICOMweb con AWS HealthImaging

Puede recuperar objetos DICOM de AWS HealthImaging mediante una representación de [DICOMweb](#) APIs, que están basados en APIs la web y siguen el estándar DICOM para imágenes médicas. [Esta funcionalidad le permite interoperar con sistemas que utilizan archivos binarios de la parte 10 de DICOM y, al mismo tiempo, aprovechar las acciones nativas de la HealthImaging nube.](#) Este capítulo se centra en cómo utilizar la implementación HealthImaging de DICOMweb APIs para devolver respuestas. DICOMweb

Importante

HealthImaging almacena los datos DICOM como [conjuntos de imágenes](#). Utilice acciones HealthImaging nativas de la nube para gestionar y recuperar conjuntos de imágenes. HealthImagingSe DICOMweb APIs pueden usar para devolver información sobre el conjunto de imágenes con respuestas DICOMweb conformes con los parámetros. Los que APIs se enumeran en este capítulo están diseñados de conformidad con el [DICOMweb](#) estándar de imágenes médicas basadas en la web. Debido a que son representaciones de DICOMweb APIs, no se ofrecen a través de AWS CLI y AWS SDKs.

Topic

- [Almacenamiento de instancias con STOW-RS](#)
- [Recuperación de datos DICOM de HealthImaging](#)
- [Búsqueda de datos DICOM en HealthImaging](#)
- [Autenticación OIDC para DICOMweb APIs](#)

Almacenamiento de instancias con STOW-RS

AWS HealthImaging ofrece una representación de los datos [DICOMweb STOW-RS](#) APIs para importar. APIs Utilícelas para almacenar de forma sincrónica los datos DICOM en su almacén de HealthImaging datos.

En la siguiente tabla se describen las HealthImaging representaciones del DICOMweb STOW-RS APIs disponibles para importar datos.

HealthImaging representaciones de STOW-RS DICOMweb APIs

Name	Description (Descripción)
StoreDICOM	Almacene una o más instancias en un banco de HealthImaging datos.
StoreDICOMStudy	Almacene una o más instancias correspondientes a un UID de instancia de estudio específico en un banco de HealthImaging datos.

Los datos importados con las StoreDICOMStudy acciones StoreDICOM y se organizarán como nuevos conjuntos de imágenes principales o se agregarán a los conjuntos de imágenes principales existentes, utilizando la misma lógica que los trabajos de importación asíncrona. Si los elementos de metadatos de los datos del DICOM P10 recién importados entran en conflicto con los conjuntos de imágenes principales existentes, los nuevos datos se añadirán a los conjuntos de imágenes no principales.

 Note

- Estas acciones permiten cargar hasta 1 GB de datos DICOM por solicitud.
- La respuesta de la API estará en formato JSON, de conformidad con el estándar STOW-RS. DICOMweb

Para iniciar una solicitud StoreDICOM

1. Recopile su región de AWS y el nombre del archivo DICOM P10. HealthImaging datastoreId
2. Cree una URL para la solicitud del formulario: `https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/studies`
3. Determine la longitud del contenido del archivo DICOM P10 mediante el comando que prefiera, por ejemplo. `$(stat -f %z $FILENAME)`
4. Prepare y envíe su solicitud. StoreDICOM utiliza una solicitud HTTP POST con el protocolo de [firma AWS Signature versión 4](#).

Example Ejemplo 1: Para almacenar un archivo DICOM P10 mediante la acción **StoreDICOM**

Shell

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

Example Ejemplo 2: Para almacenar un archivo DICOM P10 mediante la acción **StoreDICOMStudy**

La única diferencia entre StoreDICOM y Store DICOMStudy es que se pasa un UID de instancia de estudio como parámetro a Store DICOMStudy y las instancias cargadas deben ser miembros del estudio especificado.

Shell

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

Example Ejemplo 3: almacenar archivos DICOM P10 con una carga HTTP de varias partes

Se pueden cargar varios archivos P10 con una sola acción de carga de varias partes. Los siguientes comandos de shell muestran cómo ensamblar una carga útil de varias partes que contiene dos archivos P10 y cómo cargarla con la acción. StoreDICOM

Shell

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

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

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

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

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

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

# Delete the payload file
```

```
rm $FILENAME
```

Recuperación de datos DICOM de HealthImaging

AWS HealthImaging ofrece representaciones de [DICOMweb WADO-RS](#) APIs cómo recuperar datos a nivel de serie e instancia. Con ellas APIs, es posible recuperar todos los metadatos de una serie DICOM de un [almacén de HealthImaging datos](#). También es posible recuperar una instancia DICOM, los metadatos de una instancia DICOM y los marcos de una instancia DICOM (datos de píxeles). HealthImagingDICOMweb WADO-RS APIs ofrecen flexibilidad a la hora de recuperar los datos almacenados HealthImaging y proporcionan interoperabilidad con las aplicaciones antiguas.

Importante

HealthImaging almacena los datos DICOM como conjuntos de [imágenes](#). Utilice [acciones nativas de HealthImaging la nube](#) para gestionar y recuperar conjuntos de imágenes.

HealthImagingse DICOMweb APIs pueden usar para devolver información sobre el conjunto de imágenes con respuestas DICOMweb conformes con los parámetros.

Los que APIs se enumeran en esta sección están diseñados de conformidad con el estándar DICOMweb (WADO-RS) para imágenes médicas basadas en la web. Como son representaciones de DICOMweb APIs, no se ofrecen a través de y. AWS CLI AWS SDKs

En la siguiente tabla se describen todas las HealthImaging representaciones de DICOMweb WADO-RS APIs disponibles para recuperar datos. HealthImaging

HealthImaging representaciones de WADO-RS DICOMweb APIs

Name	Description (Descripción)
GetDICOMSeriesMetadata	Para recuperar los metadatos de una instancia DICOM (.jsonarchivo) de una serie DICOM en un banco de HealthImaging datos, especifique el estudio y la serie UUIDs asociados a un recurso. Consulte Recupera los metadatos de la serie .
GetDICOMInstance	Para recuperar una instancia (.dcmarchivo) de DICOM de un banco de HealthImaging datos,

Name	Description (Descripción)
	especifique la serie, el estudio y la instancia UUIDs asociados a un recurso. Consulte Recupera una instancia .
GetDICOMInstanceMetadata	Para recuperar los metadatos (.jsonarchivo) de una instancia DICOM en un banco de HealthImaging datos, especifique la serie, el estudio y la instancia UUIDs asociados a un recurso. Consulte Recuperar metadatos de instancia .
GetDICOMInstanceFrames	Recupere fotogramas de imagen individuales o por lotes (multipart solicitud) de una instancia DICOM en un almacén de HealthImaging datos especificando el UID de serie, el UID de estudio, la instancia y los números de fotograma UUIDs asociados a un recurso. Consulte Recuperar marcos .

Temas

- [Obtener una instancia DICOM de HealthImaging](#)
- [Obtener metadatos de instancias DICOM de HealthImaging](#)
- [Obtención de metadatos de la serie DICOM de HealthImaging](#)
- [Obtener marcos de instancia DICOM de HealthImaging](#)
- [Obtención de datos masivos de DICOM desde HealthImaging](#)

Obtener una instancia DICOM de HealthImaging

Utilice la GetDICOMInstance acción para recuperar una instancia (.dcmarchivo) DICOM de un [almacén de HealthImaging datos](#) especificando la serie, el estudio y la instancia UUIDs asociados al recurso. La API solo devolverá instancias de conjuntos de imágenes principales, a menos que se proporcione el [parámetro de conjunto de imágenes](#) opcional. Puede recuperar cualquier instancia (de conjuntos de imágenes principales o no principales) del almacén de datos especificándola

imageSetId como parámetro de consulta. Los datos DICOM se pueden recuperar en su sintaxis de transferencia almacenada o en formato sin comprimir (ELE).

Para obtener una instancia DICOM () **.dcm**

1. Recopile valores HealthImaging datastoreId y imageSetId parametrize.
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. Cree una URL para la solicitud con los valores de datastoreId studyInstanceUIDseriesInstanceUID, sopInstanceUID, y imageSetId. Para ver la ruta URL completa en el siguiente ejemplo, desplázate sobre el botón Copiar. La URL tiene el siguiente formato:

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/
studies/study-instance-uid/series/series-instance-uid/instances/sop-instance-uid?
imageSetId=image-set-id
```

4. Prepara y envía tu solicitud. GetDICOMInstance utiliza una solicitud HTTP GET con el protocolo de [AWS firma Signature versión 4](#). En el siguiente ejemplo de código, se utiliza la herramienta de línea de curl comandos para obtener una instancia (.dcmarchivo) DICOM. HealthImaging

Shell

```
curl --request GET \
  'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
d9a2a515ab294163a2d2f4069eed584c/
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/
instances/1.3.6.1.4.1.5962.1.1.4.1.1.20040826186059.5457?
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

El `transfer-syntax` UID es 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 para marcos de imagen sin pérdidas)
- En `transfer-syntax=*` ese caso, los fotogramas de la imagen se devolverán con la sintaxis de transferencia almacenada.
- Compresión de imagen JPEG 2000 de alto rendimiento con opciones de RPCL (solo sin pérdidas): 1.2.840.10008.1.2.4.202 - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.202
- JPEG 2000 sin pérdidas: 1.2.840.10008.1.2.4.90 - si la instancia está almacenada como sin pérdidas. HealthImaging
- Línea base de JPEG (proceso 1): sintaxis de transferencia predeterminada para la compresión de imágenes JPEG de 8 bits con pérdidas: 1.2.840.10008.1.2.4.50 - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.50
- Compresión de imagen JPEG 2000 1.2.840.10008.1.2.4.91: - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.91
- Compresión de imágenes JPEG 2000 de alto rendimiento:
1.2.840.10008.1.2.4.203 - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.203
- Compresión de imágenes JPEG XL 1.2.840.10008.1.2.4.112: - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.112
- Las instancias almacenadas HealthImaging con uno o más fotogramas de imagen codificados en la familia de [sintaxis de transferencia](#) MPEG (que incluye MPEG2 MPEG-4 AVC/H.264 and HEVC/H.265) se pueden recuperar con el UID de sintaxis de transferencia correspondiente. Por ejemplo, 1.2.840.10008.1.2.4.100 si la instancia se almacena como nivel principal del perfil principal. MPEG2

Para obtener más información, consulte [Sintaxis de transferencia compatibles](#) y [Bibliotecas de decodificación de marcos de imágenes para AWS HealthImaging](#).

Obtener metadatos de instancias DICOM de HealthImaging

Utilice la `GetDICOMInstanceMetadata` acción para recuperar los metadatos de una instancia DICOM en un [almacén de HealthImaging datos](#) especificando la serie, el estudio y la instancia UUIDs asociados al recurso. La API solo devolverá los metadatos de las instancias de los conjuntos de imágenes principales, a menos que se proporcione el parámetro de [conjunto de imágenes](#) opcional. Puede recuperar cualquier metadato de la instancia (de conjuntos de imágenes principales o no principales) del almacén de datos especificándolo `imageSetId` como parámetro de consulta.

Para obtener los metadatos de la instancia DICOM () **.json**

1. Recopile valores `HealthImaging datastoreId` y `imageSetId` paramétrice.
2. Utilice la [GetImageSetMetadata](#) acción con los valores de los `imageSetId` parámetros `datastoreId` y para recuperar los valores de metadatos asociados para `studyInstanceUIDseriesInstanceUID`, `sopInstanceUID`. Para obtener más información, consulte [Obtención de metadatos de conjuntos de imágenes](#).
3. Cree una URL para la solicitud con los valores de `datastoreId` `studyInstanceUIDseriesInstanceUID`, `sopInstanceUID`, y `imageSetId`. Para ver la ruta URL completa en el siguiente ejemplo, desplázate sobre el botón Copiar. La URL tiene el siguiente formato:

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/studies/study-instance-uid/series/series-instance-uid/instances/sop-instance-uid/metadata?imageSetId=image-set-id
```

4. Prepara y envía tu solicitud. `GetDICOMInstanceMetadata` utiliza una solicitud HTTP GET 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 (.json archivo) de HealthImaging la instancia DICOM.

Shell

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

```

```
--aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
--user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
--header "x-amz-security-token:$AWS_SESSION_TOKEN" \
--header 'Accept: application/dicom+json'
```

Note

El UID de sintaxis de transferencia indicado en los metadatos coincide con el UID de sintaxis de transferencia almacenado () StoredTransferSyntaxUID de HealthImaging

Obtención de metadatos de la serie DICOM de HealthImaging

Utilice esta GetDICOMSeriesMetadata acción para recuperar los metadatos de una serie (.jsonarchivo) DICOM de un almacén de HealthImaging [datos](#). Puede recuperar los metadatos de las series de cualquier [conjunto de imágenes](#) principal del HealthImaging banco de datos especificando el estudio y la serie UUIDs asociados al recurso. Puede recuperar los metadatos de las series de conjuntos de imágenes no principales proporcionando el ID del conjunto de imágenes como parámetro de consulta. Los metadatos de la serie se devuelven en DICOM JSON formato.

Para obtener los metadatos de la serie DICOM () **.json**

1. Recopile valores HealthImaging datastoreId y imageSetId parametrize.
2. Cree una URL para la solicitud con los valores dedatastoreId,, y studyInstanceUIDseriesInstanceUID, de forma opcionalimageSetId. Para ver la ruta URL completa en el siguiente ejemplo, desplázate sobre el botón Copiar. La URL tiene el siguiente formato:

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

3. Prepara y envía tu solicitud. GetDICOMSeriesMetadatautiliza una solicitud HTTP GET con el protocolo de [AWS firma Signature versión 4](#). En el siguiente ejemplo de código se utiliza la herramienta de línea de curl comandos para obtener metadatos (.jsonarchivos) HealthImaging.

Shell

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

Con el `imageSetId` parámetro opcional.

Shell

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

Note

- El `imageSetId` parámetro es necesario para recuperar los metadatos de las series de conjuntos de imágenes no principales. La `GetDICOMInstanceMetadata` acción solo devolverá los metadatos de la serie para los conjuntos de imágenes principales si `seriesInstanceUID` se especifican los valores `datastoreIdstudyInstanceUID`, (sin `unimagesetID`).

Obtener marcos de instancia DICOM de HealthImaging

Utilice esta `GetDICOMInstanceFrames` acción para recuperar fotogramas de imagen individuales o por lotes (`multipartolicitud`) de una instancia DICOM en un [almacén de HealthImaging datos](#) especificando el UID de serie, el UID de estudio, la instancia UIDs y los números de fotograma asociados a un recurso. Puede especificar el [conjunto de imágenes](#) del que se deben recuperar los marcos de la instancia proporcionando el ID del conjunto de imágenes como parámetro de consulta. La API solo devolverá fotogramas de instancias de los conjuntos de imágenes principales, a menos que se proporcione el parámetro de [conjunto de imágenes](#) opcional. Puede recuperar cualquier marco de instancia (de conjuntos de imágenes principales o no principales) del almacén de datos especificándolo `imageSetId` como parámetro de consulta.

Los datos DICOM se pueden recuperar en su sintaxis de transferencia almacenada o en formato sin comprimir (ELE).

Para obtener marcos de instancia DICOM () **multipart**

1. Recopile valores `HealthImaging datastoreId` y `imageSetId` paramétrice.
2. Utilice la [GetImageSetMetadata](#) acción con los valores de los `imageSetId` parámetros `datastoreId` y para recuperar los valores de metadatos asociados para `studyInstanceUID`, `seriesInstanceUID`, `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 fotograma de imagen de los metadatos será el fotograma 1.
 - Solicitud de fotograma único: `/frames/1`
 - Solicitud de fotogramas múltiples: `/frames/1,2,3,4`
4. Crea una URL para la solicitud con los valores de `datastoreId`, `studyInstanceUID`, `seriesInstanceUID`, `sopInstanceUID`, `imageSetId`, y `frameList`. Para ver la ruta URL completa en el siguiente ejemplo, desplázate sobre el botón Copiar. La URL tiene el siguiente formato:

```
GET https://dicom-medical-imaging.region.amazonaws.com/datastore/datastore-id/studies/study-instance-uid/series/series-instance-uid/instances/sop-instance-uid/frames/1?imageSetId=image-set-id
```

5. Prepara y envía tu solicitud. `GetDICOMInstanceFrames` utiliza una solicitud HTTP GET 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 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

El `transfer-syntax` UID es opcional y, si no se incluye, el valor predeterminado es Explicit VR Little Endian. Si la transcodificación a ELE no es posible (debido a una importación con advertencia), los píxeles se devolverán sin transcodificarlos. Las sintaxis de transferencia compatibles incluyen:

- Explicit VR Little Endian (ELE): 1.2.840.10008.1.2.1 (predeterminado para marcos de imagen sin pérdidas)
- En `transfer-syntax=*` ese caso, los fotogramas de la imagen se devolverán con la sintaxis de transferencia almacenada.
- Compresión de imagen JPEG 2000 de alto rendimiento con opciones de RPCL (solo sin pérdidas): 1.2.840.10008.1.2.4.202 - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.202
- JPEG 2000 sin pérdidas: 1.2.840.10008.1.2.4.90 - si la instancia está almacenada como sin pérdidas. HealthImaging

- Línea base de JPEG (proceso 1): sintaxis de transferencia predeterminada para la compresión de imágenes JPEG de 8 bits con pérdidas: 1.2.840.10008.1.2.4.50 - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.50
- Compresión de imagen JPEG 2000 1.2.840.10008.1.2.4.91: - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.91
- Compresión de imágenes JPEG 2000 de alto rendimiento:
1.2.840.10008.1.2.4.203 - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.203
- Compresión de imágenes JPEG XL 1.2.840.10008.1.2.4.112: - si la instancia está almacenada en HealthImaging 1.2.840.10008.1.2.4.112
- Las instancias almacenadas HealthImaging con uno o más fotogramas de imagen codificados en la familia de [sintaxis de transferencia](#) MPEG (que incluye MPEG2 MPEG-4 AVC/H.264 and HEVC/H.265) se pueden recuperar con el UID de sintaxis de transferencia correspondiente. Por ejemplo, 1.2.840.10008.1.2.4.100 si la instancia se almacena como nivel principal del perfil principal. MPEG2
- Es posible que reciba un 406 NotAcceptableException si la sintaxis de transferencia solicitada no se puede devolver en función de la sintaxis de transferencia almacenada o si hay advertencias de procesamiento específicas para la instancia. Si esto ocurre, vuelva a intentar la llamada con `transfer-syntax=*`.

Para obtener más información, consulte [Sintaxis de transferencia compatibles](#) y [Bibliotecas de decodificación de marcos de imágenes para AWS HealthImaging](#).

Obtención de datos masivos de DICOM desde HealthImaging

Utilice la `GetDICOMBulkdata` acción para recuperar datos binarios que se hayan separado de los metadatos DICOM en un banco de HealthImaging datos. Al recuperar metadatos de instancias o series, los atributos binarios de más de 1 MB se representarán mediante a `BulkDataURI` en lugar de valores en línea. Puede recuperar los datos binarios de cualquier conjunto de imágenes principal del almacén de HealthImaging datos utilizando los datos `BulkDataURI` proporcionados en la respuesta de metadatos. Puede recuperar datos masivos de conjuntos de imágenes no principales proporcionando el ID del conjunto de imágenes como parámetro de consulta.

Para obtener datos masivos de DICOM

Al recuperar metadatos DICOM de una acción de HealthImaging DICOMweb WADO-RS (por ejemplo, GetDICOMInstanceMetadata o), los atributos binarios grandes se sustituirán en línea por GetDICOMSeriesMetadata, como se muestra a continuación: BulkData URIs

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

Para recuperar un elemento DICOM con la GetDICOMBulkdata acción, sigue estos pasos.

1. Cree una URL para la solicitud utilizando los valores del BulkDataURI, del formulario:

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

2. Emita su GetDICOMBulkdata comando como una solicitud HTTP GET con el protocolo de [firma AWS Signature versión 4](#). El siguiente ejemplo de código utiliza la herramienta de línea de `curl` comandos para recuperar un elemento DICOM de un conjunto de imágenes principal:

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

Para recuperar un elemento de datos DICOM de un conjunto de imágenes no principal, introduzca un parámetro: ImageSetId

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

```

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

```

Note

El `imageSetId` parámetro es necesario para recuperar datos masivos de conjuntos de imágenes no principales. La DICOMBulkdata acción Obtener solo devolverá datos masivos de los conjuntos de imágenes principales si se `SOPInstanceUID` especifican los valores `datastoreId`, `studyInstanceUID` `seriesInstanceUID`, y (sin ninguno). `imagesetID`

Búsqueda de datos DICOM en HealthImaging

AWS HealthImaging ofrece representaciones de [DICOMweb QIDO-RS](#) APIs para buscar estudios, series e instancias por ID de paciente y recibir sus identificadores únicos para su uso posterior. HealthImagingLos DICOMweb QIDO-RS APIs ofrecen flexibilidad a la hora de buscar los datos almacenados en las aplicaciones antiguas HealthImaging y ofrecen interoperabilidad con ellas.

Importante

HealthImagingse DICOMweb APIs pueden utilizar para devolver información del conjunto de imágenes con el QIDO-RS. HealthImaging DICOMweb APIs hacen referencia únicamente a [conjuntos de imágenes](#) a menos que se indique lo contrario. Utilice [las acciones nativas de la HealthImaging nube](#) o el parámetro opcional de DICOMweb las acciones del conjunto de imágenes para recuperar conjuntos de imágenes no principales. HealthImagingLos s se DICOMweb APIs pueden usar para devolver información sobre el conjunto de imágenes con respuestas DICOMweb conformes con -.

HealthImaging DICOMweb Las acciones de QIDO-RS pueden devolver un máximo de 10 000 registros. [En el caso de que existan más de 10 000 recursos, no se podrán recuperar](#)

[mediante las acciones de QIDO-RS, pero se podrán recuperar mediante las acciones de DICOMweb WADO-RS o las acciones nativas de la nube.](#)

Los que APIs se enumeran en esta sección están diseñados de conformidad con el estándar DICOMweb (QIDO-RS) para imágenes médicas basadas en la web. No se ofrecen a través de y. AWS CLI AWS SDKs

DICOMweb APIs buscar HealthImaging

En la siguiente tabla se describen todas las HealthImaging representaciones de DICOMweb QIDO-RS APIs disponibles para buscar datos en. HealthImaging

HealthImaging representaciones de QIDO-RS DICOMweb APIs

Name	Description (Descripción)
SearchDICOMStudies	Busque estudios DICOM especificando los elementos de HealthImaging la consulta de búsqueda mediante una solicitud GET. Los resultados de la búsqueda de estudios se muestran en formato JSON, ordenados por última actualización y por fecha descendente (de la más reciente a la más antigua). Consulte Busque estudios .
SearchDICOMSeries	Busque series DICOM HealthImaging especificando los elementos de la consulta de búsqueda mediante una solicitud GET. Los resultados de la búsqueda de series se muestran en formato JSON, ordenados Series Number (0020, 0011) en orden ascendente (del más antiguo al más reciente). Consulte Busque series .
SearchDICOMInstances	Busque instancias DICOM HealthImaging especificando los elementos de la consulta de búsqueda mediante una solicitud GET. Los resultados de la búsqueda de instancia s se muestran en formato JSON, ordenados

Name	Description (Descripción)
	Instance Number (0020, 0013) en orden ascendente (del más antiguo al más reciente). Consulte Busque instancias .

Tipos de DICOMweb consulta compatibles para HealthImaging

HealthImaging admite consultas jerárquicas de recursos QIDO-RS en los niveles de estudio, serie e instancia SOP. Cuando utilice el QIDO-RS, busque jerárquicamente: HealthImaging

- La búsqueda de estudios devuelve una lista de estudios
- La búsqueda de una serie de estudios requiere una serie conocida StudyInstanceUID y devuelve una lista de series
- La búsqueda en una lista de instancias requiere un dato conocido StudyInstanceUID y SeriesInstanceUID

En la siguiente tabla se describen los tipos de consultas jerárquicas QIDO-RS compatibles para buscar datos. HealthImaging

HealthImaging tipos de consultas QIDO-RS compatibles

Tipo de consulta	Ejemplo
Consultas de valores de atributos	Busque todas las series de un estudio dondemodality=CT . <pre>.../studies/1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894866/series?00080060=CT</pre> Busque todos los estudios en los que la identificación del paciente y la fecha del estudio sean estos valores, respectivamente. <pre>.../studies?PatientID=11235813&StudyDate=20130509</pre>

Tipo de consulta	Ejemplo
Consultas de palabras clave	Busque en todas las series con la SeriesInstanceUID palabra clave. <pre>.../studies/1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894866/series?SeriesInstanceUID=1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894868</pre>
Etiquetas: consultas	Busque etiquetas mediante los parámetros de consulta introducidos en el group/element formulario. {grupo} {elemento} como 0020000D
Consultas de rango	<pre>...?Modality=CT&StudyDate=ABBYYYY-BBCCYYYY</pre>
Paginación de resultados con y limit offset	<pre>.../studies?limit=1&offset=0&00080020=20000101</pre> Puede utilizar los parámetros de límite y desplazamiento para paginar las respuestas de búsqueda. El valor predeterminado del límite es 1000 y consulte HealthImaging Puntos de enlace y cuotas de AWS para obtener el valor máximo. Límite máximo = 1000, desfase máximo = 9000

Tipo de consulta	Ejemplo
Consultas comodín	Las consultas con caracteres comodín proporcionan más flexibilidad en la búsqueda mediante «*» y «?». «*» coincide con cualquier secuencia de caracteres (incluido un valor de longitud cero) y «?» coincide con cualquier carácter individual. Busque todos los estudios en un almacén de datos que StudyDescription contenga «Nuclear»: <pre>.../studies?StudyDescription=*Nuclear*</pre> Busque todos los estudios que StudyDescription terminen en «Nuclear»: <pre>.../studies?StudyDescription=*Nuclear</pre> Busque todos los estudios que StudyDescription comiencen por «Nuclear»: <pre>.../studies?StudyDescription=Nuclear*</pre> Busque todos los estudios en los que el identificador de paciente tenga exactamente 3 caracteres cualquiera después de 200965981: <pre>.../studies?PatientID=200965981???</pre>

Tipo de consulta	Ejemplo
FuzzyMatching consultas	Habilite la coincidencia parcial en los atributos DICOM de los nombres (PatientName (0010,0010), ReferringPhysicianName (0008,0090)) añadiendo el parámetro de consulta opcional fuzzymatching: <pre>.../studies?fuzzymatching=true&PatientName="Thomas^Albert"</pre> Esta consulta realiza coincidencias de palabras de prefijo que no distingan mayúsculas de minúsculas en cualquier parte del valor. PatientName Devuelve resultados con PatientName valores como «thomas», «Albert», «Thomas Albert», «Thomas^Albert», pero no «hom» ni «ber».

Temas

- [Búsqueda de estudios DICOM en HealthImaging](#)
- [Búsqueda de series DICOM en HealthImaging](#)
- [Búsqueda de instancias DICOM en HealthImaging](#)

Búsqueda de estudios DICOM en HealthImaging

[Utilice la SearchDICOMStudies API para buscar estudios DICOM en un almacén de datos.](#)

[HealthImaging](#) Para buscar estudios DICOM, cree una URL que incluya los elementos de datos DICOM compatibles (atributos). HealthImaging Los resultados de la búsqueda de estudios se muestran en formato JSON, ordenados por última actualización y por fecha descendente (de la última a la más antigua).

Para buscar estudios DICOM

1. HealthImaging regionRecopila y datastoreId valora. Para obtener más información, consulte [Obtención de propiedades de los almacenes de datos](#).

2. Cree una URL para la solicitud, que incluya todos los elementos del estudio aplicables. Para ver la ruta URL completa en el siguiente ejemplo, desplázate sobre el botón Copiar. La URL tiene el siguiente formato:

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

Elementos de estudio para **SearchDICOMStudies**

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

3. Prepara y envía tu solicitud. SearchDICOMStudies utiliza una solicitud HTTP GET con el protocolo de [AWS firma Signature versión 4](#). En el siguiente ejemplo, se utiliza la herramienta de línea de `curl` comandos para buscar información sobre los estudios del DICOM.

curl

```
curl --request GET \
  "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
  studies[?query]"
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom+json' \
  --output results.json
```

Los resultados de la búsqueda de estudios se devuelven en formato JSON, ordenados por última actualización y por fecha descendente (de la última a la más antigua).

Búsqueda de series DICOM en HealthImaging

[Utilice la SearchDICOMSeries API para buscar series DICOM en un almacén de datos.](#)

[HealthImaging](#) Para buscar series DICOM, cree una URL que incluya los elementos de datos DICOM compatibles (atributos). HealthImaging Los resultados de la búsqueda de series se muestran en formato JSON, ordenados de forma ascendente (del más antiguo al más reciente).

Para buscar series DICOM

1. HealthImaging regionRecopila y datastoreId valora. Para obtener más información, consulte [Obtención de propiedades de los almacenes de datos](#).
2. Recoge el StudyInstanceUID valor. Para obtener más información, consulte [Obtención de metadatos de conjuntos de imágenes](#).
3. Crea una URL para la solicitud, que incluya todos los elementos de la serie aplicables. Para ver la ruta URL completa en el siguiente ejemplo, desplázate sobre el botón Copiar. La URL tiene el siguiente formato:

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

Elementos de serie para **SearchDICOMSeries**

Etiqueta de elemento DICOM	Nombre del elemento DICOM
(0008,0060)	Modality
(0020,000E)	Series Instance UID

4. Prepara y envía tu solicitud. SearchDICOMSeries utiliza una solicitud HTTP GET con el protocolo de [AWS firma Signature versión 4](#). En el siguiente ejemplo, se utiliza la herramienta de línea de `curl` comandos para buscar información sobre la serie DICOM.

`curl`

```
curl --request GET \
  "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
  studies/StudyInstanceUID/series[?query]"
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom+json' \
  --output results.json
```

Los resultados de la búsqueda de series se devuelven en formato JSON, ordenados Series Number (0020,0011) en orden ascendente (de la más antigua a la más reciente).

Búsqueda de instancias DICOM en HealthImaging

[Utilice la SearchDICOMInstances API para buscar instancias DICOM en un almacén de datos.](#)

[HealthImaging](#) Para buscar instancias de DICOM, cree una URL que incluya los elementos de datos DICOM (atributos) compatibles. HealthImaging Los resultados de la instancia se devuelven en formato JSON, ordenados de forma ascendente (del más antiguo al más reciente).

Para buscar instancias DICOM

1. HealthImaging regionRecopila y datastoreId valora. Para obtener más información, consulte [Obtención de propiedades de los almacenes de datos](#).

2. Recopila valores para `StudyInstanceUID` y `SeriesInstanceUID`. Para obtener más información, consulte [Obtención de metadatos de conjuntos de imágenes](#).
3. Cree una URL para la solicitud, que incluya todos los elementos de búsqueda aplicables. Para ver la ruta URL completa en el siguiente ejemplo, desplázate sobre el botón Copiar. La URL tiene el siguiente formato:

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

Elementos de instancia para **SearchDICOMInstances**

Etiqueta de elemento DICOM	Nombre del elemento DICOM
(0008,0016)	SOP Class UID
(0008,0018)	SOP Instance UID
(0008,1196)	WarningReason

HealthImaging utiliza el elemento DICOM [\(0008,1196\)](#) para conservar los códigos de advertencia de importación. Los códigos de advertencia de importación se pueden buscar a nivel de instancia. Los códigos de advertencia de importación se pueden buscar con caracteres comodín o códigos de advertencia específicos. Consulte [HealthImaging Códigos de advertencia](#).

4. Prepara y envía tu solicitud. `SearchDICOMInstances` utiliza una solicitud HTTP GET con el protocolo de [AWS firma Signature versión 4](#). En el siguiente ejemplo, se utiliza la herramienta de línea de `curl` comandos para buscar información sobre las instancias DICOM.

`curl`

```
curl --request GET \
  "https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/datastoreId/
  studies/StudyInstanceUID/series/SeriesInstanceUID/instances[?query]"
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom+json' \
  --output results.json
```

Los resultados de la búsqueda de instancias se devuelven en formato JSON, ordenados Instance Number (0020,0013) en orden ascendente (de la más antigua a la más reciente)

Autenticación OIDC para DICOMweb APIs

AWS HealthImaging admite la autenticación basada en [OAuth 2.0](#) para las solicitudes de DICOMweb API mediante [OpenID Connect \(OIDC\)](#), además de la autenticación [AWS Signature Version 4](#) (Sigv4) existente. El OIDC le permite integrarse HealthImaging directamente con proveedores de identidad externos (IdPs) y permite que las aplicaciones basadas en estándares accedan a sus datos de imágenes médicas a través de puntos de conexión sin necesidad de que cada aplicación tenga credenciales HealthImaging DICOMweb . AWS

Temas

- [Verificación de token personalizada con autorizadores Lambda](#)
- [Configurar un autorizador AWS Lambda para la autenticación OIDC](#)

Verificación de token personalizada con autorizadores Lambda

HealthImaging implementa el soporte de OIDC a través de una arquitectura que utiliza autorizadores Lambda, lo que permite a los clientes implementar su propia lógica de verificación de token. Este diseño le permite controlar de forma flexible cómo se validan los tokens y cómo se aplican las decisiones de acceso, y se adapta a un panorama diverso de proveedores de identidad compatibles con el OIDC () y a distintos métodos de verificación de los tokens. IdPs

Flujo de autenticación

Así es como funciona la autenticación a un alto nivel:

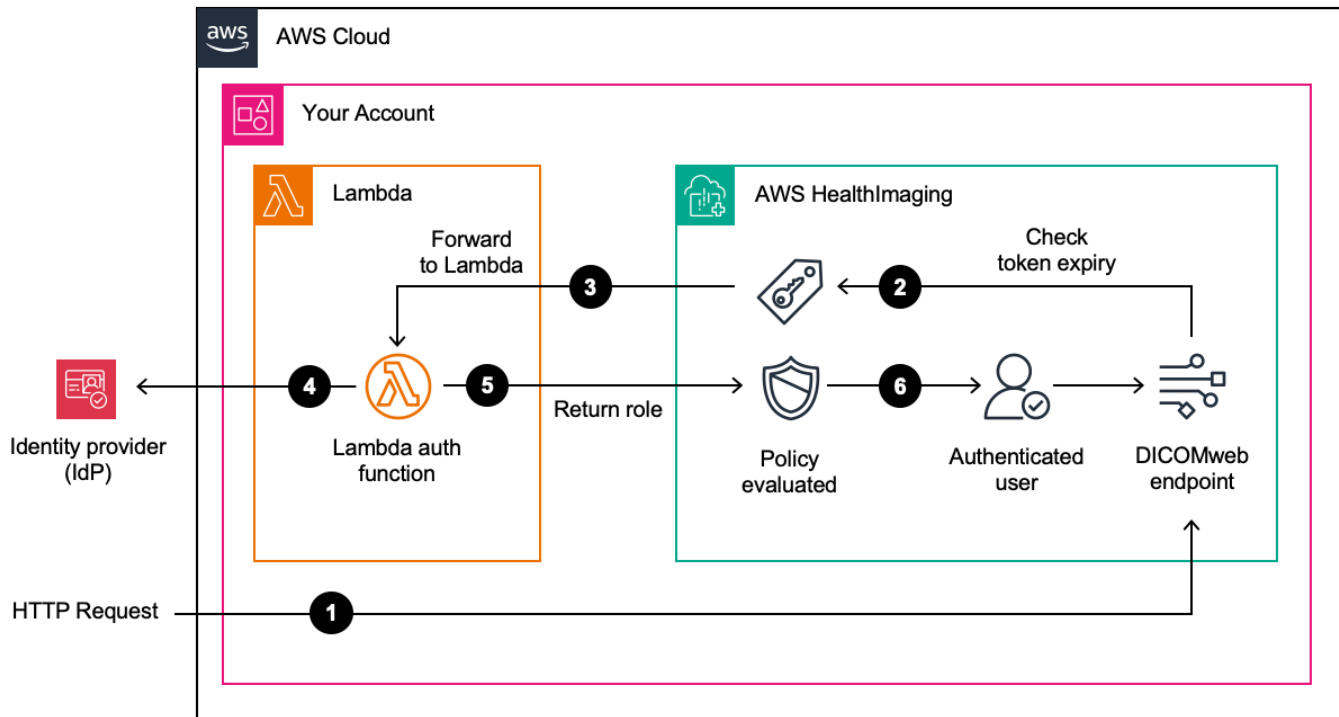
1. El cliente llama a la DICOMweb API: tu aplicación se autentica con el proveedor de identidad OIDC que hayas elegido y recibe un token de identificación firmado (JWT). Para cada solicitud DICOMweb HTTP, el cliente debe incluir el token de acceso OIDC en el encabezado de autorización (normalmente, un token de portador). Antes de que la solicitud llegue a sus datos, HealthImaging extrae este token de la solicitud entrante y llama al autorizador Lambda que usted configure.
 - a. El encabezado suele seguir el formato: `Authorization: Bearer <token>`

2. Verificación inicial: HealthImaging verifica las solicitudes de token de acceso para rechazar rápidamente cualquier token obviamente inválido o caducado sin invocar innecesariamente la función Lambda. HealthImaging realiza una verificación inicial de determinadas afirmaciones estándar en el token de acceso antes de invocar el autorizador Lambda:
 - a. `iat`(Emitido el): HealthImaging comprueba si el tiempo de emisión del token está dentro de los límites aceptables.
 - b. `exp`(Hora de caducidad): HealthImaging verifica que el token no haya caducado.
 - c. `nbf`(No antes de la hora): si está presente, HealthImaging garantiza que el token no se esté utilizando antes de la hora de inicio válida.
3. HealthImaging invoca un autorizador de Lambda: si se aprueba la verificación de la reclamación inicial HealthImaging , delega la verificación de token adicional a la función de autorización de Lambda configurada por el cliente. HealthImaging pasa el token extraído y otra información de solicitud relevante a la función Lambda. La función Lambda verifica la firma y las afirmaciones del token.
4. Verificar con un proveedor de identidad: La Lambda contiene un código personalizado que comprueba la firma del token de ID, realiza una verificación del token más exhaustiva (por ejemplo, del emisor, la audiencia o las reclamaciones personalizadas) y valida esas reclamaciones contra el IdP cuando es necesario.
5. El autorizador devuelve una política de acceso: tras una verificación correcta, la función Lambda determina los permisos adecuados para el uso autenticado. A continuación, el autorizador Lambda devuelve el nombre del recurso de Amazon (ARN) de una función de IAM que representa el conjunto de permisos que se van a conceder.
6. Ejecución de la solicitud: si la función de IAM asumida tiene los permisos necesarios, HealthImaging procede a devolver el recurso solicitado. DICOMWeb Si los permisos son insuficientes, HealthImaging deniega la solicitud y devuelve el error de respuesta correspondiente (es decir, 403 Prohibido).

Note

El servicio AWS HealthImaging no administra la función lambda del autorizador. Se ejecuta en su cuenta. AWS A los clientes se les cobra por el tiempo de invocación y ejecución de la función por separado de sus HealthImaging cargos.

Información general de la arquitectura



Flujo de trabajo de autenticación OIDC con autorizador Lambda

Requisitos previos

Requisitos del token de acceso

HealthImaging requiere que el token de acceso esté en formato JSON Web Token (JWT). Muchos proveedores de identidad (IDPs) ofrecen este formato de token de forma nativa, mientras que otros permiten seleccionar o configurar el formulario de token de acceso. Asegúrese de que el IDP que elija pueda emitir los tokens JWT antes de continuar con la integración.

Formato de token

El token de acceso debe estar en formato JWT (JSON Web Token)

Reclamaciones requeridas

`exp`(Hora de caducidad)

Declaración obligatoria que especifica cuándo deja de ser válido el token.

- Debe ser posterior a la hora actual en UTC
- Representa el momento en que el token deja de ser válido

`iat`(Emitido el)

Declaración obligatoria que especifica cuándo se emitió el token.

- Debe ser anterior a la hora actual en UTC
- NO debe ser anterior a 12 horas antes de la hora actual en UTC
- De hecho, esto impone una vida útil máxima del token de 12 horas

`nbf`(No antes de tiempo)

Declaración opcional que especifica la primera vez que se puede usar el token.

- Si está presente, será evaluado por HealthImaging
- Especifica el tiempo antes del cual no se debe aceptar el token

Requisitos de tiempo de respuesta del autorizador Lambda

HealthImaging impone estrictos requisitos de temporización para las respuestas de los autorizadores Lambda a fin de garantizar un rendimiento óptimo de la API. La función Lambda debe volver en 1 segundo.

Prácticas recomendadas

Optimice la verificación del token

- Guarde en caché los JWKS (conjuntos de claves web JSON) cuando sea posible
- Siempre que sea posible, guarde en caché los tokens de acceso válidos
- Minimice las llamadas de red a su proveedor de identidad
- Implemente una lógica de validación de tokens eficiente

Configuración Lambda

- Las funciones basadas en Python y Node.js normalmente se inicializan más rápido
- Reduzca la cantidad de bibliotecas externas que se van a cargar
- Configure la asignación de memoria adecuada para garantizar un rendimiento uniforme
- Supervise los tiempos de ejecución mediante CloudWatch métricas

Habilitación de la autenticación OIDC

- La autenticación OIDC solo se puede habilitar al crear un nuevo almacén de datos
- La API no admite la activación de OIDC para los almacenes de datos existentes
- Para habilitar OIDC en un almacén de datos existente, los clientes deben ponerse en contacto con Support AWS

Configurar un autorizador AWS Lambda para la autenticación OIDC

Esta guía asume que ya ha configurado el proveedor de identidad (IdP) de su elección para proporcionar tokens de acceso compatibles con los requisitos de la función de autenticación del HealthImaging OIDC.

1. Configure las funciones de IAM para el acceso a la API DICOMWeb

Antes de configurar el autorizador Lambda, cree las funciones de IAM para que las asuma HealthImaging al procesar las solicitudes de API. DICOMWeb La función Lambda del autorizador devuelve el ARN de uno de estos roles tras una verificación correcta del token, HealthImaging lo que permite ejecutar las solicitudes con los permisos adecuados.

1. Cree políticas de IAM que definan los privilegios de API deseados. DICOMWeb Consulte la sección «[Uso DICOMweb](#)» de la HealthImaging documentación para ver los permisos disponibles.
2. Cree funciones de IAM que:
 - Adjunte estas políticas
 - Incluya una relación de confianza que permita al director de HealthImaging servicio de AWS (`medical-imaging.amazonaws.com`) asumir estas funciones.

A continuación, se muestra un ejemplo de una política que permite a los roles asociados acceder a una API de HealthImaging DICOMWeb solo lectura:

A continuación, se muestra un ejemplo de la política de relaciones de confianza que debe asociarse a los roles:

JSON

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

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

El autorizador Lambda que creará en el siguiente paso puede evaluar las reclamaciones de token y devolver el ARN de la función correspondiente. A continuación, AWS HealthImaging suplantarán esta función para ejecutar la solicitud de DICOMWeb API con los permisos correspondientes.

Por ejemplo:

- Un token con afirmaciones de «administrador» puede devolver un ARN para un rol con acceso total
- Un token con afirmaciones de «lector» puede devolver un ARN para un rol con acceso de solo lectura
- Un token con la leyenda «Department_A» puede devolver un ARN para un rol específico del nivel de acceso de ese departamento.

Este mecanismo le permite asignar el modelo de autorización de su IdP a HealthImaging permisos específicos de AWS mediante funciones de IAM.

2. Creación y configuración de la función de autorización Lambda

Cree una función Lambda que verifique el token JWT y devuelva el ARN del rol de IAM correspondiente en función de la evaluación de las afirmaciones del token. El servicio de imágenes médicas invoca esta función y pasa un evento que contiene el identificador del HealthImaging almacén de datos, la DICOMWeb operación y el token de acceso que se encuentran en la solicitud HTTP:

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

```
"bearerToken": "{access token}"
}
```

La función autorizadora de Lambda debe devolver una respuesta JSON con la siguiente estructura:

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

Puede consultar el ejemplo de implementación para obtener más información.

Note

Como la DICOMWeb solicitud solo se responde después de que el autorizador lambda verifique el token de acceso, es importante que la ejecución de esta función sea lo más rápida posible para proporcionar el mejor tiempo de respuesta de la DICOMWeb API.

Para que el HealthImaging servicio esté autorizado a invocar la función de autorización lambda, debe tener una política de recursos que permita HealthImaging al servicio invocarla. Esta política de recursos se puede crear en el menú de permisos de la pestaña de configuración de lambda o mediante: AWS CLI

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

Esta política de recursos permite que el HealthImaging servicio invoque su autorizador Lambda al DICOMWeb autenticar las solicitudes de API.

Note

La política de recursos de lambda se puede actualizar más adelante con una condición «ArnLike» que coincida con el ARN de un HealthImaging almacén de datos específico.

A continuación, se muestra un ejemplo de política de recursos de lambda:

3. Cree un nuevo almacén de datos con autenticación OIDC

Para habilitar la autenticación OIDC, debe crear un nuevo almacén de datos con el parámetro "». AWS CLI `lambda-authorizer-arn` La autenticación OIDC no se puede habilitar en los almacenes de datos existentes sin ponerse en contacto con Support. AWS

A continuación, se muestra un ejemplo de cómo crear un almacén de datos nuevo con la autenticación OIDC habilitada:

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

Puede comprobar si un almacén de datos específico tiene habilitada la función de autenticación OIDC mediante el comando AWS CLI `get-datastore` y comprobando si el atributo `lambdaAuthorizerArn` está presente:

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

```
{  
  "datastoreProperties": {  
    "datastoreId": YourdatastoreId,  
    "datastoreName": YourDatastoreName,  
    "datastoreStatus": "ACTIVE",  
    "lambdaAuthorizerArn": YourAuthorizerFunctionArn,  
    "datastoreArn": YourDatastoreArn,  
    "createdAt": "2025-09-30T14:16:04.015000-05:00",  
    "updatedAt": "2025-09-30T14:16:04.015000-05:00"  
  }  
}
```

Note

La función de ejecución del comando de creación del AWS CLI almacén de datos debe tener los permisos adecuados para invocar la función autorizadora de Lambda. Esto mitiga los ataques de escalada de privilegios en los que usuarios malintencionados podrían ejecutar funciones Lambda no autorizadas a través de la configuración del autorizador del almacén de datos.

Códigos de excepción

En caso de error de autenticación, HealthImaging devuelve los siguientes códigos de respuesta al error HTTP y mensajes del cuerpo:

Condición	Respuesta AHI
El autorizador Lambda no existe o no es válido	424 Authorizer está mal configurado
El autorizador se canceló debido a un error de ejecución	424 Falló el autorizador
¿Algún otro error de autorizador o no mapeado	Error en el autorizador 424
El autorizador devolvió una respuesta inválida o mal formada	424 Configuración errónea del autorizador
Authorizer ejecutó más de 1 segundo	Tiempo de espera del autorizador 408
El token ha caducado o no es válido	403 El token no es válido o ha caducado
AHI no puede federar el rol de IAM devuelto debido a una mala configuración del autorizador	424 Configuración errónea del autorizador
El autorizador devolvió un rol vacío	403 Acceso denegado
El rol devuelto no es invocable (error de configuración de assume-role/trust)	424 Configuración errónea del autorizador

Condición	Respuesta AHI
La tasa de solicitudes supera DICOMweb los límites de Gateway	429 Demasiadas solicitudes
Almacén de datos, rol de devolución o autorizador entre regiones Account/Cross	424 Autoriza el acceso entre regiones Account/Cross

Ejemplo de implementación

En este ejemplo de Python, se muestra una función autorizadora lambda que verifica los tokens de acceso de AWS Cognito a HealthImaging partir de eventos y devuelve un ARN de rol de IAM con los privilegios adecuados. DICOMWeb

El autorizador Lambda implementa dos mecanismos de almacenamiento en caché para reducir las llamadas externas y la latencia de respuesta. El JWKS (conjunto de claves web JSON) se obtiene una vez cada hora y se almacena en la carpeta temporal de la función, lo que permite que las invocaciones posteriores a la función lo lean localmente en lugar de obtenerlo desde la red pública. También observará que se crea una instancia de un objeto de diccionario token_cache en el contexto global de esta función Lambda. Todas las invocaciones que reutilizan el mismo contexto Lambda calentado comparten las variables globales. Gracias a esto, los tokens verificados correctamente se pueden almacenar en este diccionario y consultarlos rápidamente durante la próxima ejecución de esta misma función Lambda. El método de almacenamiento en caché representa un enfoque generalista que podría adaptarse a los tokens de acceso emitidos por la mayoría de los proveedores de identidad. [Para ver una opción de almacenamiento en caché específica de AWS Cognito, consulte la sección Gestión del grupo de usuarios y la sección de almacenamiento en caché de la documentación de Cognito.AWS](#)

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

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

# Global token cache with TTL
token_cache = {}

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

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

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

def get_cached_jwks():
    """Get JWKS from cache file if valid, otherwise return None """
    try:
        if os.path.exists(JWKS_CACHE_FILE):
            # Check if cache file is still valid
            cache_age = time.time() - os.path.getmtime(JWKS_CACHE_FILE)
            if cache_age < JWKS_CACHE_TTL:
                with open(JWKS_CACHE_FILE, 'r') as f:
                    jwks = json.load(f)
                    logger.debug(f'Using cached JWKS (age: {int(cache_age)}s)')
                    return jwks
            else:
                logger.debug(f'JWKS cache expired (age: {int(cache_age)}s)')
    except Exception as e:
        logger.debug(f'Error reading JWKS cache: {e}')

    return None
```

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

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

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

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

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

    return cached

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

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

def handler(event, context):
```

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

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

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

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

    # Decode token header to get kid
    headers = jwt.get_unverified_headers(token)
    kid = headers['kid']

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

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

```
        break

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

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

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

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

    # Cache the validated token
    cache_token(token, payload)

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

except ExpiredSignatureError:
    logger.debug('Token expired')
    return {
        'isTokenValid': False,
        'roleArn': ''
    }

except JWTClaimsError:
    logger.debug('Invalid token claims')
    return {
        'isTokenValid': False,
        'roleArn': ''
    }

except JWTErrors as e:
    logger.debug('JWT validation error: %s', e)
    return {
```

```
        'isTokenValid': False,  
        'roleArn': ''  
    }  
except Exception as e:  
    logger.debug('Authorization failed: %s', e)  
    return {  
        'isTokenValid': False,  
        'roleArn': ''  
    }
```

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 HealthImaging datos](#) de AWS. Durante la importación, AWS HealthImaging realiza una [comprobación de los datos de píxeles](#) antes de transformar los archivos DICOM P10 en [conjuntos de imágenes](#) compuestos por [metadatos](#) y [marcos de imágenes](#) (datos de píxeles).

Importante

HealthImaging Los trabajos de importación procesan .dcm archivos binarios de instancias DICOM y los transforman en conjuntos de imágenes. Utilice [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 describe cómo importar los datos de imágenes médicas a un banco de HealthImaging datos mediante Consola de administración de AWS AWS CLI, y AWS SDKs.

Temas

- [Introducción a los trabajos de importación](#)
- [Inicio de un trabajo de importación](#)
- [Obtención de las propiedades de trabajos de importación](#)
- [Enumeración de trabajos de importación](#)

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 campos Consola de administración de AWS AWS CLI, y AWS SDKs para iniciar, describir y enumerar los trabajos de importación.

[Al importar los datos de DICOM P10 a un almacén de datos de AWS HealthImaging , el servicio intenta organizar automáticamente las instancias según la jerarquía DICOM de UID de estudio, UID de serie y UID de instancia, en función de los elementos de metadatos.](#) Los datos importados

pasarán a ser principales si los [elementos de metadatos de los](#) datos importados no entran en conflicto con los conjuntos de [imágenes](#) principales existentes en el almacén de datos. [Si los elementos de metadatos de los datos del DICOM P10 recién importados entran en conflicto con los conjuntos de imágenes principales existentes, los nuevos datos se añadirán a los conjuntos de imágenes no principales.](#) Cuando las importaciones de datos crean [conjuntos de imágenes](#) no principales, AWS HealthImaging emite un EventBridge evento con `isPrimary: False` el objeto y el registro escrito en él también `success.ndjson` estará `isPrimary: False` dentro del `importResponse` objeto.

Al importar datos, HealthImaging hace lo siguiente:

- Si las instancias que componen una serie DICOM se importan en un trabajo de importación y las instancias no entran en conflicto con las que ya se encuentran en el banco de datos, todas las instancias se organizan en un [conjunto de imágenes](#) principal.
- Si las instancias que componen una serie DICOM se importan en dos o más trabajos de importación y las instancias no entran en conflicto con las que ya se encuentran en el banco de datos, todas las instancias se organizan como un conjunto de [imágenes](#) principal.
- Si una instancia se importa más de una vez, la última versión sobrescribirá cualquier versión anterior almacenada en un [conjunto de imágenes](#) principal y se incrementará el número de versión del [conjunto de imágenes](#) principal.

Puedes actualizar las instancias de la instancia principal siguiendo los pasos que se describen en [Actualización de los metadatos del conjunto de imágenes](#).

Durante la importación, los valores binarios de las etiquetas privadas (con los tipos VR OB, OD, OF, OL, OV, OW, UN) que superen el tamaño de 1 MB se almacenan por separado de los metadatos. Al recuperar los metadatos de estas instancias mediante `GetDICOMInstanceMetadata` o `GetDICOMSeriesMetadata`, estos valores binarios grandes se sustituyen por `BulkDataURIs`, y los datos binarios reales se pueden recuperar mediante la `GetDICOMBulkdata` API.

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

- Las instancias correspondientes a una serie DICOM se combinarán automáticamente en un único conjunto de imágenes, denominado primario.
- Puede importar los datos del DICOM P10 en un trabajo de importación o en varios trabajos de importación, y el servicio organizará las instancias en conjuntos de imágenes principales que correspondan a la serie DICOM

- 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 [Restricciones de los elementos DICOM](#).
- 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 limitación 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/",
    "warningsOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/WARNING/",
```

```

        "numberOfScannedFiles": 5,
        "numberOfImportedFiles": 3,
        "numberOfFilesWithCustomerError": 2,
        "numberOfFilesWithServerError": 0,
        "numberOfGeneratedImageSets": 2,
        "imageSetsSummary": [{
"imageSetId": "12345612345612345678907890789012",
            "numberOfMatchedSOPInstances": 2
        },
        {
"imageSetId": "12345612345612345678917891789012",
            "numberOfMatchedSOPInstances": 1
        }
    ]
}
}

```

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

```

- 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"}}

```

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

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

- 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

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

Note

Tenga en cuenta los siguientes puntos antes de iniciar un trabajo de importación:

- HealthImaging admite la importación de archivos DICOM P10 con diferentes sintaxis de transferencia. Algunos archivos conservan su codificación de sintaxis de transferencia original durante la importación, mientras que otros se transcodifican a HTJ2 K lossless de forma predeterminada o a JPEG 2000 Lossless, según la configuración del almacén de datos. Para obtener más información, consulte [Sintaxis de transferencia compatibles](#).
- 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](#).
- HealthImaging aplica restricciones de longitud a elementos DICOM específicos durante la importación. Para obtener más información, consulte [Restricciones de los elementos DICOM](#).

Los menús siguientes proporcionan un procedimiento Consola de administración de AWS y ejemplos de código para las teclas AWS CLI y AWS SDKs. Para obtener más información, consulte [StartDICOMImportJob](#) la referencia de la HealthImaging API de AWS.

Inicio de un trabajo 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.
3. Seleccione Importar datos DICOM.

Se abrirá la página de Importar datos DICOM.

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++

SDK para C++

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

```

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

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

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

```

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

- Para obtener más información sobre la API, consulta [Start DICOMImport Job](#) in AWS SDK para C++ API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta Cómo [iniciar un trabajo de importación](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener información sobre la API, consulte [Start DICOMImport Job](#) in AWS CLI Command Reference.

Java

SDK para Java 2.x

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

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

    return "";
}
```

- Para obtener más información sobre la API, consulta [Start DICOMImport Job](#) in AWS SDK for Java 2.x API Reference.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

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

- Para obtener más información sobre la API, consulta [Start DICOMImport Job](#) in AWS SDK para JavaScript API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param job_name: The name of the job.
        :param datastore_id: The ID of the data store.
```

```

        :param role_arn: The Amazon Resource Name (ARN) of the role to use for
the job.
        :param input_s3_uri: The S3 bucket input prefix path containing the DICOM
files.
        :param output_s3_uri: The S3 bucket output prefix path for the result.
        :return: The job ID.
    """
    try:
        job = self.health_imaging_client.start_dicom_import_job(
            jobName=job_name,
            datastoreId=datastore_id,
            dataAccessRoleArn=role_arn,
            inputS3Uri=input_s3_uri,
            outputS3Uri=output_s3_uri,
        )
    except ClientError as err:
        logger.error(
            "Couldn't start DICOM import job. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return job["jobId"]

```

El siguiente código crea una instancia del `MedicalImagingWrapper` objeto.

```

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

```

- Para obtener más información sobre la API, consulta la referencia de la API [Start DICOMImport Job](#) in AWS SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
  " iv_job_name = 'import-job-1'
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_role_arn = 'arn:aws:iam::123456789012:role/ImportJobRole'
  " iv_input_s3_uri = 's3://my-bucket/input/'
  " iv_output_s3_uri = 's3://my-bucket/output/'
  oo_result = lo_mig->startdicomimportjob(
    iv_jobname = iv_job_name
    iv_datastoreid = iv_datastore_id
    iv_dataaccessrolearn = iv_role_arn
    iv_inputs3uri = iv_input_s3_uri
    iv_outputs3uri = iv_output_s3_uri ).
  DATA(lv_job_id) = oo_result->get_jobid( ).
  MESSAGE |DICOM import job started with ID: { lv_job_id }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
  MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [Start DICOMImport Job](#) in AWS SDK para obtener una referencia sobre la API ABAP de SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace [Enviar comentarios](#) en la barra lateral derecha de esta página.

Obtención de las propiedades de trabajos de importación

Utilice esta `GetDICOMImportJob` acción para obtener más información sobre las propiedades de los trabajos de HealthImaging importación de AWS. 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 siguientes menús proporcionan un procedimiento Consola de administración de AWS y ejemplos de código para el AWS CLI y AWS SDKs. Para obtener más información, consulte [GetDICOMImportJob](#) la referencia de la HealthImaging API de AWS.

Cómo obtener las propiedades de los 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.

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++

SDK para C++

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

    return outcome;
}
```

- Para obtener más información sobre la API, consulta [Get DICOMImport Job](#) in AWS SDK para C++ API Reference.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta [Cómo obtener propiedades de trabajos de importación](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener información sobre la API, consulte [Get DICOMImport Job](#) in AWS CLI Command Reference.

Java

SDK para Java 2.x

```
public static DICOMImportJobProperties getDicomImportJob(MedicalImagingClient
medicalImagingClient,
                String datastoreId,
                String jobId) {

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

    return null;
}
```

- Para obtener más información sobre la API, consulta [Get DICOMImport Job](#) in AWS SDK for Java 2.x API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //     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/
  output_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxx'/',
  //     submittedAt: 2023-09-19T17:27:25.143Z
  //   }
  // }

  return response;
};

```

- Para obtener más información sobre la API, consulta [Get DICOMImport Job](#) in AWS SDK para JavaScript API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para [obtener más información sobre la API, consulta la referencia de la API Get DICOMImport Job](#) in AWS SDK for Python (Boto3).

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_job_id = '12345678901234567890123456789012'
  oo_result = lo_mig->getdicomimportjob(
    iv_datastoreid = iv_datastore_id
    iv_jobid = iv_job_id ).
  DATA(lo_job_props) = oo_result->get_jobproperties( ).
  DATA(lv_job_status) = lo_job_props->get_jobstatus( ).
  MESSAGE |Job status: { lv_job_status }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Job not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
```

```
ENDTRY.
```

- Para [obtener más información sobre la API, consulte la referencia Get DICOMImport Job](#) in AWS SDK para la API ABAP de SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Enumeración de trabajos de importación

Utilice la `ListDICOMImportJobs` acción para enumerar los trabajos de importación creados para un [almacén HealthImaging de datos](#) específico. Los siguientes menús proporcionan un procedimiento Consola de administración de AWS y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte [ListDICOMImportJobs](#) la referencia de la HealthImaging API de AWS.

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 la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en la referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

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

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

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en la referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

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

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

```

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

return jobSummaries;
};
```

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en la referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :return: The list of jobs.
        """
        try:
            paginator = self.health_imaging_client.get_paginator(
                "list_dicom_import_jobs"
            )
            page_iterator = paginator.paginate(datastoreId=datastore_id)
            job_summaries = []
            for page in page_iterator:
```

```

        job_summaries.extend(page["jobSummaries"])
    except ClientError as err:
        logger.error(
            "Couldn't list DICOM import jobs. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return job_summaries

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta la [sección Lista de DICOMImport trabajos](#) en la referencia de la API del AWS SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->listdicomimportjobs( iv_datastoreid =
iv_datastore_id ).
    DATA(lt_jobs) = oo_result->get_jobsummaries( ).
    DATA(lv_count) = lines( lt_jobs ).
    MESSAGE |Found { lv_count } DICOM import jobs.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.

```

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

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en el AWS SDK para obtener una referencia sobre la API ABAP de SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Acceso a conjuntos de imágenes con AWS HealthImaging

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

Importante

Durante la importación, HealthImaging procesa los .dcm archivos binarios de las instancias DICOM y los transforma en conjuntos de imágenes. Utilice [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 Consola de administración de AWS 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](#)
- [Búsqueda de conjuntos de imágenes](#)
- [Obtención de las propiedades del conjunto de imágenes](#)
- [Obtención de metadatos de conjuntos de imágenes](#)
- [Obtención de datos de píxeles de los conjuntos de imágenes](#)

Descripción de los conjuntos de imágenes

Los conjuntos de imágenes AWS se parecen a los de la serie DICOM y son la base de AWS HealthImaging. Los conjuntos de imágenes se crean al importar los datos DICOM a. HealthImaging El servicio intenta organizar los datos P10 importados según la jerarquía DICOM de estudio, serie e instancia.

Los conjuntos de imágenes se introdujeron por varias razones:

- Support una amplia variedad de flujos de trabajo de imágenes médicas (clínicos y no clínicos) mediante la flexibilidad APIs.
- Proporcione un mecanismo para almacenar y conciliar de forma duradera los datos duplicados e incoherentes. Los datos P10 importados que entren en conflicto con los conjuntos de imágenes principales que ya estén en un almacén se conservarán como datos no principales. Tras resolver los conflictos de metadatos, esos datos pueden pasar a ser principales.
- 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 que se parece mucho a los de la serie DICOM. 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](#). En otras palabras, un conjunto común de atributos y valores corresponde a los elementos a nivel de paciente, estudio y serie que figuran en el [Registro de elementos de datos DICOM](#). HealthImaging utiliza los siguientes elementos DICOM al agrupar los objetos DICOM P10 entrantes en conjuntos de imágenes.

Elementos DICOM utilizados para la creación de conjuntos de imágenes

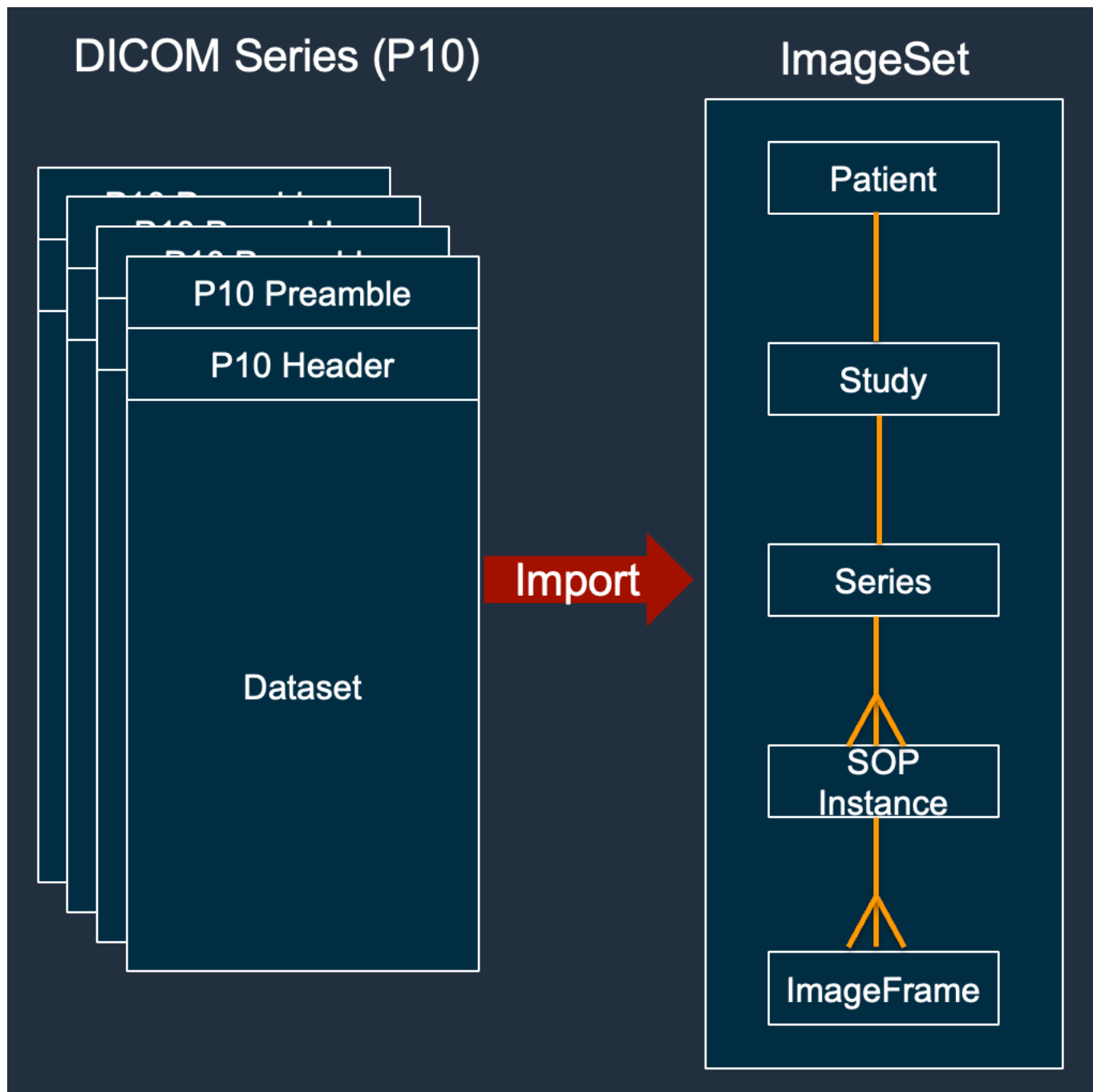
Nombre del elemento	Etiqueta de elemento
Elementos a nivel de estudio	
Study Date	(0008,0020)
Accession Number	(0008,0050)
Patient ID	(0010,0020)
Study Instance UID	(0020,000D)
Study ID	(0020,0010)
Elementos a nivel de serie	
Series Instance UID	(0020,000E)
Series Number	(0020,0011)

Durante la importación, algunos conjuntos de imágenes conservan su codificación de sintaxis de transferencia original, mientras que otros se transcodifican a JPEG 2000 (HTJ2K) de alto rendimiento sin pérdidas de forma predeterminada. Si un conjunto de imágenes está codificado en HTJ2 K, debe decodificarse antes de visualizarlo. Para obtener más información, consulte [Sintaxis de transferencia compatibles](#) y [Bibliotecas de decodificación de marcos de imágenes](#).

Los fotogramas de imagen (datos en píxeles) están codificados en JPEG 2000 (HTJ2K) de alto rendimiento y deben [decodificarse](#) antes de visualizarlos.

Los conjuntos de imágenes son AWS recursos, por lo que se les asignan [nombres de recursos de Amazon \(ARNs\)](#). 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 o incremente la versión de los conjuntos de imágenes existentes.
- No deduplique el almacenamiento de instancias SOP. Cada importación de la misma instancia de SOP utiliza almacenamiento adicional como un nuevo conjunto de imágenes no principal o una versión incrementada de un conjunto de imágenes principal existente.
- Organice automáticamente las instancias SOP con metadatos consistentes y no conflictivos como conjuntos de imágenes principales, que contienen instancias con elementos de metadatos consistentes de pacientes, estudios y series.
 - Si las instancias que componen una serie DICOM se importan en dos o más trabajos de importación y las instancias no entran en conflicto con las que ya se encuentran en el almacén de datos, todas las instancias se organizarán en un conjunto de imágenes principal.
- Cree conjuntos de imágenes no principales que contengan datos del DICOM P10 que entren en conflicto con los conjuntos de imágenes principales que ya están en el almacén de datos.
- Conserve los datos recibidos más recientemente como la última versión de un conjunto de imágenes principal.
 - Si las instancias que componen una serie DICOM son conjuntos de imágenes principales y se vuelve a importar una instancia, la nueva copia se insertará en el conjunto de imágenes principal y se incrementará la versión.

¿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": {
```

```

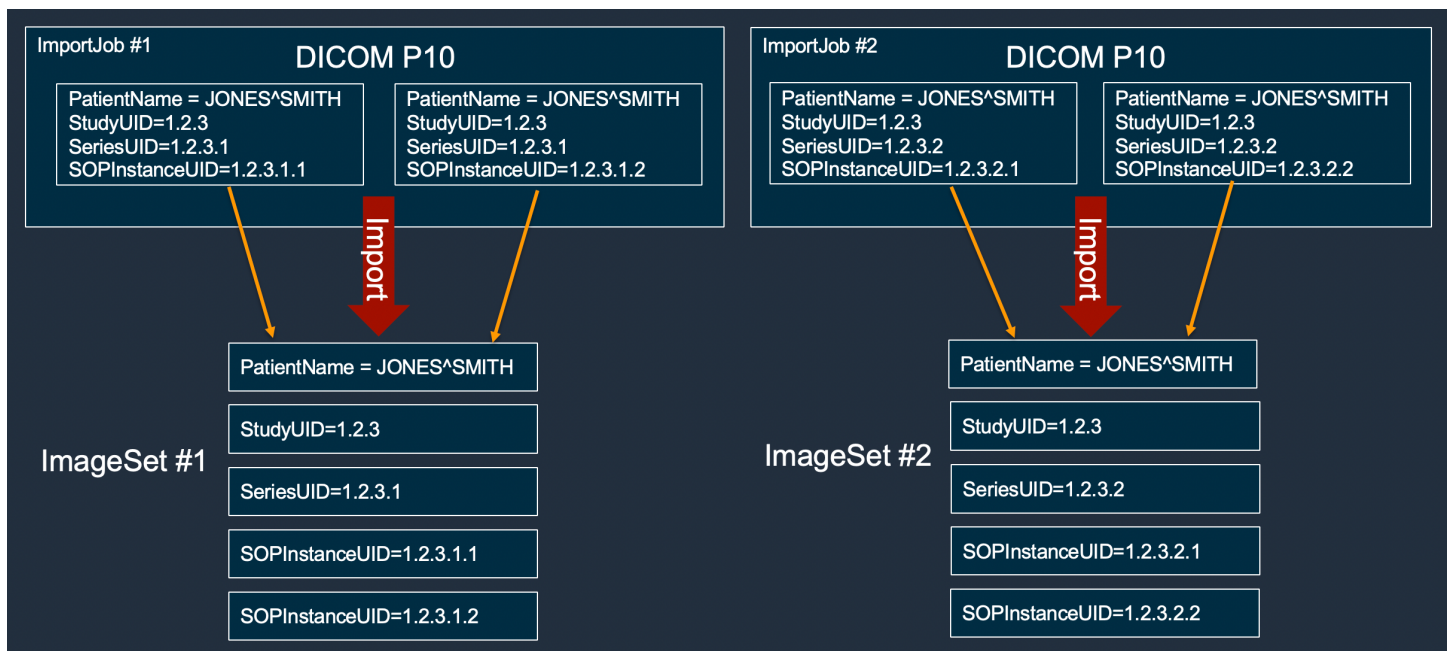
    "PatientBirthDate": null,
    "PatientSex": null,
    "PatientID": "2178309",
    "PatientName": "MISTER^CT"
  }
},
"Study": {
  "DICOM": {
    "StudyTime": "083501",
    "PatientWeight": null
  },
  "Series": {
    "1.2.840.113619.2.30.1.1762295590.1623.978668949.887": {
      "DICOM": {
        "Modality": "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
          }
        }
      }
    }
  }
}

```

```
}
]
}
}
}
}
}
```

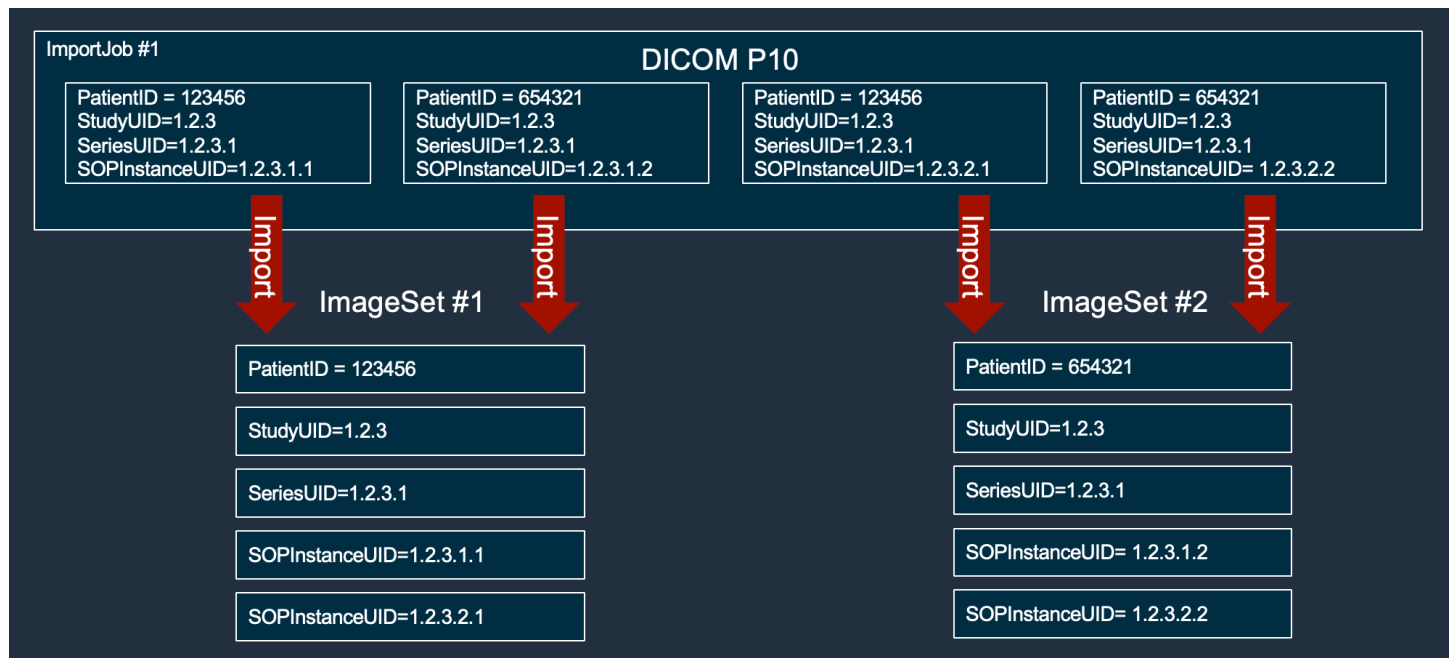
Ejemplo de creación de un conjunto de imágenes: varios trabajos de importación

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



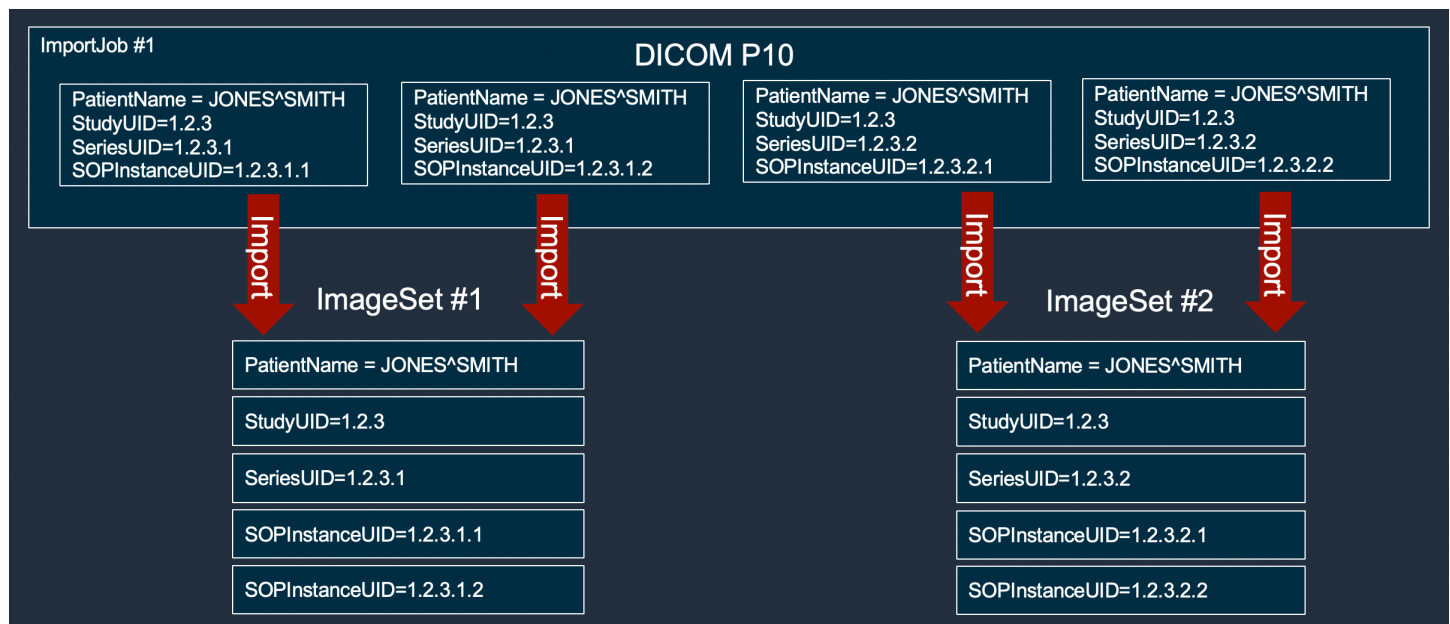
Ejemplo de creación de un conjunto de imágenes: trabajo de importación único con dos variantes

El siguiente ejemplo muestra un único trabajo de importación que no se combinaría en un único conjunto de imágenes porque las instancias 1 y 3 tienen un paciente IDs diferente al de las instancias 2 y 4. Para resolver este problema, puede utilizar la `UpdateImageSetMetadata` acción para resolver el conflicto de identificación del paciente con el conjunto de imágenes principal existente. Una vez resueltos los conflictos, puede utilizar la `CopyImageSet` acción con el argumento `--promoteToPrimary` para añadir el conjunto de imágenes al conjunto de imágenes principal.



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. En los menús siguientes se proporciona un procedimiento Consola de administración de AWS y ejemplos de código para AWS CLI y AWS SDKs. Para obtener más información, consulte [SearchImageSets](#) la referencia de la HealthImaging API de AWS.

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 el almacén de datos con una AWS KMS clave propiedad del cliente, debe actualizar la política de AWS KMS claves antes de interactuar con los 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) el UID de instancia de serie que le interese.

Note

Los valores del UID de instancia de serie deben ser idénticos a los que figuran en el [registro de identificadores únicos DICOM \(\)](#). 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 serie UID. No se permiten letras ni espacios en blanco, así que tenga cuidado al copiar y pegar UIDs.

5. Seleccione el menú Intervalo de fechas, seleccione un intervalo de fechas para el UID de instancia de 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++

SDK para 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 núm. 1: operador IGUAL.

```

        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: el operador BETWEEN usa DICOMStudy fecha y DICOMStudy hora.

```

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

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

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

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

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

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

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

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

        if (result) {
            std::cout << usesCase2Results.size() << " image sets found for
between 1999/01/01 and present."

```

```

        << std::endl;
        for (auto &imageSetResult : usesCase2Results) {
            std::cout << "    Image set with ID '" << imageSetResult <<
std::endl;
        }
    }
}

```

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```

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

    useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T000000000Z", Aws::Utils::DateSpec::RFC3339));

    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 obtener más información sobre la API, consulta la Referencia de la API.
[SearchImageSets](#) AWS SDK para C++

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

CLI

AWS CLI

Ejemplo 1: búsqueda de conjuntos de imágenes con un operador EQUAL

En el siguiente ejemplo de código `search-image-sets` se usa el operador EQUAL 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 operador BETWEEN mediante DICOMStudy fecha y DICOMStudy hora

En el siguiente ejemplo de código `search-image-sets` se buscan conjuntos de imágenes con estudios DICOM generados entre el 1 de enero de 1990 (00:00 h) y el 1 de enero de 2023 (00:00 h).

Nota: La DICOMStudy hora 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: búsqueda de conjuntos de imágenes con un operador BETWEEN mediante `createdAt` (los estudios de tiempo se conservaban previamente)

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

Nota: Ingrese `createdAt` en el formato de ejemplo ("1985-04-12T23:20:50.52Z").

```
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 operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordenar la respuesta en orden ASC en el campo UpdatedAt

El siguiente ejemplo de search-image-sets código busca conjuntos de imágenes con un operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

Nota: Introduzca updatedAt en el formato de ejemplo ("1985-04-12T23:20:50.52Z").

```

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, consulta la sección Búsqueda de conjuntos de imágenes en la Guía para desarrolladores.AWS HealthImaging](#)

- Para obtener más información sobre la API, consulte [SearchImageSets](#) la Referencia de AWS CLI comandos.

Java

SDK para 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 núm. 1: operador IGUAL.

```

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: ENTRE el operador que usa DICOMStudy fecha y DICOMStudy hora.

```

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMdd");
        searchFilters = Collections.singletonList(SearchFilter.builder()
            .operator(Operator.BETWEEN)
            .values(SearchByAttributeValue.builder()

                .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                    .dicomStudyDate("19990101")
                    .dicomStudyTime("000000.000")
                    .build())
                .build(),
            SearchByAttributeValue.builder()

                .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                    .dicomStudyDate(LocalDate.now()
                        .format(formatter))
                    .dicomStudyTime("000000.000")
                    .build())
                .build())
            .build());

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

        imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
    datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println(
                "The image sets searched with BETWEEN operator using
DICOMStudyDate and DICOMStudyTime are:\n"
                    +
                        imageSetsMetadataSummaries);
            System.out.println();
        }
    }

```

```
}
```

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

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

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

```

        .build())
        .build(),
        SearchFilter.builder()
            .operator(Operator.BETWEEN)
            .values(

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

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

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

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

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

```

- Para obtener más información sobre la API, consulta la Referencia de la API.
[SearchImageSets](#) AWS SDK for Java 2.x

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para 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 núm. 1: operador IGUAL.

```
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: el operador BETWEEN usa DICOMStudy fecha y DICOMStudy hora.

```
const datastoreId = "12345678901234567890123456789012";

try {
```

```
const searchCriteria = {
  filters: [
    {
      values: [
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "19900101",
            DICOMStudyTime: "000000",
          },
        },
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "20230901",
            DICOMStudyTime: "000000",
          },
        },
      ],
      operator: "BETWEEN",
    },
  ],
};

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

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```
const datastoreId = "12345678901234567890123456789012";  
  
try {  
  const searchCriteria = {  
    filters: [  
      {  
        values: [  
          { updatedAt: new Date("1985-04-12T23:20:50.52Z") },  
          { updatedAt: new Date() },  
        ],  
        operator: "BETWEEN",  
      },  
      {  
        values: [  
          {  
            DICOMSeriesInstanceUID:  
              "1.1.123.123456.1.12.1.1234567890.1234.12345678.123",  
          },  
        ],  
        operator: "EQUAL",  
      },  
    ],  
    sort: {  
      sortOrder: "ASC",  
      sortField: "updatedAt",  
    },  
  };  
  
  await searchImageSets(datastoreId, searchCriteria);  
} catch (err) {  
  console.error(err);  
}
```

```
}
```

- Para obtener más información sobre la API, consulta la Referencia de la API.
[SearchImageSets](#) AWS SDK para JavaScript

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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 núm. 1: operador IGUAL.

```

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: el operador BETWEEN usa DICOMStudy fecha y DICOMStudy hora.

```

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

```

```

        ],
    }
]

image_sets = self.search_image_sets(data_store_id, search_filter)
print(
    f"Image sets found with BETWEEN operator using DICOMStudyDate and\n{image_sets}"
)

```

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```

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

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

```

MedicalImagingWrapper El siguiente código crea una instancia del objeto.

```

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

```

- Para obtener más información sobre la API, consulta [SearchImageSets](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->searchimagesets(
        iv_datastoreid = iv_datastore_id
        io_searchcriteria = io_search_criteria ).
    DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).
    DATA(lv_count) = lines( lt_imagesets ).
    MESSAGE |Found { lv_count } image sets.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [SearchImageSets](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

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 el Consola de administración de AWS y ejemplos de código para el AWS CLI y AWS SDKs. Para obtener más información, consulte [GetImageSet](#) la referencia de la HealthImaging API de AWS.

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 (`.dcm`archivo) de una instancia DICOM. Para obtener más información, consulte [Obtener una instancia DICOM 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 la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetImageSet](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

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

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

    return null;
}
```

- Para obtener más información sobre la API, consulte [GetImageSet](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

```
// }

return response;
};
```

- Para obtener más información sobre la API, consulte [GetImageSet](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

```

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

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [GetImageSet](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    " iv_version_id = '1' (optional)
    IF iv_version_id IS NOT INITIAL.
        oo_result = lo_mig->getimageset(
            iv_datastoreid = iv_datastore_id

```

```
        iv_imagesetid = iv_image_set_id
        iv_versionid = iv_version_id ).
    ELSE.
        oo_result = lo_mig->getimageset(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id ).
    ENDIF.
    DATA(lv_state) = oo_result->get_imagesetstate( ).
    MESSAGE |Image set retrieved with state: { lv_state }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [GetImageSet](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Obtención de metadatos de conjuntos de imágenes

Utilice la `GetImageSetMetadata` acción para recuperar [los metadatos](#) de un [conjunto de imágenes](#) determinado HealthImaging. Los siguientes menús proporcionan un procedimiento para el Consola de administración de AWS y ejemplos de código para el AWS CLI y AWS SDKs. Para obtener más información, consulte [GetImageSetMetadata](#) la referencia de la HealthImaging API de AWS.

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 `gzip` y se devuelven como un objeto JSON. Por lo tanto, debe descomprimir el objeto JSON antes de ver los metadatos normalizados. Para obtener más información, consulte [Normalización de metadatos](#).

Si los metadatos de un conjunto de imágenes de gran tamaño aún se están procesando después de la importación, es `ConflictException` posible que se devuelva un 409. Vuelva a intentar la solicitud transcurridos unos segundos una vez que se complete el procesamiento.

Utilice `GetDICOMInstanceMetadata` HealthImaging la representación de un DICOMweb servicio para devolver los metadatos de la instancia DICOM (.jsonarchivo). Para obtener más información, consulte [Obtener metadatos de instancias DICOM 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++

SDK para 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 obtener más información sobre la API, consulte [GetImageSetMetadata](#) la referencia AWS SDK para C++ de la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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 objeto JSON 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 objeto JSON devuelto, primero debe descomprimirlo.

Salida:

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

Para obtener más información, consulta [Cómo obtener metadatos de conjuntos de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetImageSetMetadata](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

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

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

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

```

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

```

- Para obtener más información sobre la API, consulte [GetImageSetMetadata](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para 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 "node:fs";

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

```

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

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

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

return response;
};
```

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 obtener más información sobre la API, consulte [GetImageSetMetadata](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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 MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [GetImageSetMetadata](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    " iv_version_id = '1' (optional)
    IF iv_version_id IS NOT INITIAL.
        oo_result = lo_mig->getimagesetmetadata(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id
            iv_versionid = iv_version_id ).
    ELSE.
        oo_result = lo_mig->getimagesetmetadata(
            iv_datastoreid = iv_datastore_id

```

```

        iv_imagesetid = iv_image_set_id ).
    ENDIF.
    DATA(lv_metadata_blob) = oo_result->get_imagesetmetadatablob( ).
    MESSAGE 'Image set metadata retrieved.' TYPE 'I'.
    CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
    CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
    CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
    CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
    CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
    CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.

```

- Para obtener más información sobre la API, consulte [GetImageSetMetadata](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Transfiere metadatos de sintaxis

Al importar datos DICOM, HealthImaging mantiene el valor original del atributo de sintaxis de transferencia en los metadatos del conjunto de imágenes. La sintaxis de transferencia de los datos DICOM originales importados se almacena como. TransferSyntaxUID HealthImaging se utiliza StoredTransferSyntaxUID para indicar el formato utilizado para codificar los datos de los marcos de imagen en el almacén de datos: 1.2.840.10008.1.2.4.202 para los almacenes de

datos habilitados para HTJ2 K (predeterminado) y 1.2.840.10008.1.2.4.90 para los almacenes de datos JPEG 2000 habilitados para Lossless.

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 esta `GetImageFrame` acción para recuperar un marco de imagen JPEG 2000 nativo o HTJ2 codificado en K sin pérdidas para un conjunto de [imágenes](#) determinado. HealthImaging En los menús siguientes se proporcionan ejemplos de código para las letras y. AWS CLI AWS SDKs Para obtener más información, consulte [GetImageFrame](#) la referencia de la HealthImaging API de AWS.

Note

Tenga en cuenta los siguientes puntos al utilizar la `GetImageFrame` acción:

- Durante la [importación](#), HealthImaging conserva la codificación de algunas sintaxis de transferencia y transcodifica otras a HTJ2 K lossless (predeterminado) o JPEG 2000 Lossless. La `GetImageFrame` acción devuelve el marco de la imagen en la sintaxis de transferencia almacenada de la instancia. No se realiza ninguna transcodificación durante la recuperación para garantizar una latencia de recuperación mínima. Es posible que sea necesario decodificar los fotogramas de imagen antes de visualizarlos en un visor de imágenes, según la sintaxis de transferencia. Para obtener más información, consulte [Sintaxis de transferencia compatibles](#) y [Bibliotecas de decodificación de marcos de imágenes](#).
- [En el caso de los casos almacenados HealthImaging con uno o más fotogramas de imagen codificados en la familia de sintaxis de transferencia MPEG \(que incluye MPEG-4 AVC/H.264 and HEVC/H.265\) MPEG2, la `GetImageFrame` acción devolverá un objeto de vídeo con la sintaxis de transferencia almacenada.](#)
- La sintaxis de transferencia de los fotogramas de imagen se especifica en el elemento de respuesta del encabezado. `Content-Type` HTTP Por ejemplo, un marco de imagen codificado en HTJ2 K. tendrá `Content-Type: image/jph header`. Para obtener más información, consulte [GetImageFrame](#) la referencia de la HealthImaging API de AWS.
- También puede utilizar `GetDICOMInstanceFrames` HealthImaging la representación de un DICOMweb servicio para recuperar marcos de instancias DICOM (multipartolicitud)

para aplicaciones y visores DICOMweb compatibles. Para obtener más información, consulte [Obtener marcos de instancia DICOM 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.

Consola de AWS

Note

Los marcos de imágenes deben decodificarse y se debe acceder a ellos mediante programación, ya que no hay un visor de imágenes disponible en la Consola de administración de AWS.

Para más información sobre la decodificación y la visualización de marcos de imágenes, consulte [Bibliotecas de decodificación de marcos de imágenes](#).

AWS CLI y SDKs

C++

SDK para C++

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

```
Aws::MedicalImaging::MedicalImagingClient client(clientConfig);

Aws::MedicalImaging::Model::GetImageFrameRequest request;
request.SetDatastoreId(dataStoreID);
request.SetImageSetId(imageSetID);

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

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

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

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

return outcome.IsSuccess();
}
```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la Referencia AWS SDK para C++ de la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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.jpg
```

Nota: Este ejemplo de código no incluye la salida porque la `GetImageFrame` acción devuelve un flujo de datos de píxeles al archivo `imageframe.jpg`. Para obtener información sobre la decodificación y la visualización de marcos de imágenes, consulte las bibliotecas de decodificación K. HTJ2

Para obtener más información, consulte [Obtener datos de píxeles de conjuntos de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

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

```

        .imageFrameInformation(ImageFrameInformation.builder()

        .imageFrameId(imageFrameId)

        .build())

        .build();

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreId - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.

```

```

*/
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

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

```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [GetImageFrame](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  " iv_image_set_id = '1234567890123456789012345678901234567890'
  " iv_image_frame_id = '1234567890123456789012345678901234567890'
  oo_result = lo_mig->getimageframe(
    iv_datastoreid = iv_datastore_id
    iv_imagesetid = iv_image_set_id
    io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(
      iv_imageframeid = iv_image_frame_id ) ).
  DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).
  MESSAGE 'Image frame retrieved.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Image frame not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

 Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Modificación de conjuntos de imágenes con AWS HealthImaging

Los trabajos de importación de DICOM suelen requerir que modifique sus [conjuntos de imágenes](#) por una serie de motivos:

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

Importante

Durante la importación, HealthImaging procesa los binarios (.dcmarchivos) de las instancias DICOM y los transforma en conjuntos de imágenes. Utilice [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](#)
- [Actualización de los metadatos de un conjunto de imágenes](#)
- [Copia de conjuntos de imágenes](#)
- [Eliminación de un conjunto de imágenes](#)

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 el Consola

de administración de AWS y ejemplos de código para el AWS CLI y AWS SDKs. Para obtener más información, consulte [ListImageSetVersions](#) la referencia de la HealthImaging API de AWS.

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, consulta la sección sobre [la lista de versiones de conjuntos de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

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

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

    return null;
}
```

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la Referencia AWS SDK for Java 2.x de la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

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

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

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

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
```

```
:param image_set_id: The ID of the image set.
:return: The list of image set versions.
"""
try:
    paginator = self.health_imaging_client.get_paginator(
        "list_image_set_versions"
    )
    page_iterator = paginator.paginate(
        imageSetId=image_set_id, datastoreId=datastore_id
    )
    image_set_properties_list = []
    for page in page_iterator:
        image_set_properties_list.extend(page["imageSetPropertiesList"])
except ClientError as err:
    logger.error(
        "Couldn't list image set versions. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return image_set_properties_list
```

El siguiente código crea una instancia del `MedicalImagingWrapper` objeto.

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

- Para obtener más información sobre la API, consulta [ListImageSetVersions](#) la AWS Referencia de API de SDK for Python (Boto3).

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.  
    " iv_datastore_id = '1234567890123456789012345678901234567890'  
    " iv_image_set_id = '1234567890123456789012345678901234567890'  
    oo_result = lo_mig->listimagesetversions(  
        iv_datastoreid = iv_datastore_id  
        iv_imagesetid = iv_image_set_id ).  
    DATA(lt_versions) = oo_result->get_imagesetpropertieslist( ).  
    DATA(lv_count) = lines( lt_versions ).  
    MESSAGE |Found { lv_count } image set versions.| TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
    MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Image set not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

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 seguir este proceso asíncrono para añadir, actualizar y eliminar los atributos de metadatos de los conjuntos de imágenes, que son manifestaciones de los [elementos de normalización DICOM](#) que se crean durante la importación. Con la opción `UpdateImageSetMetadata` también puede eliminar series e instancias de SOP 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 referencia de la HealthImaging API de AWS.

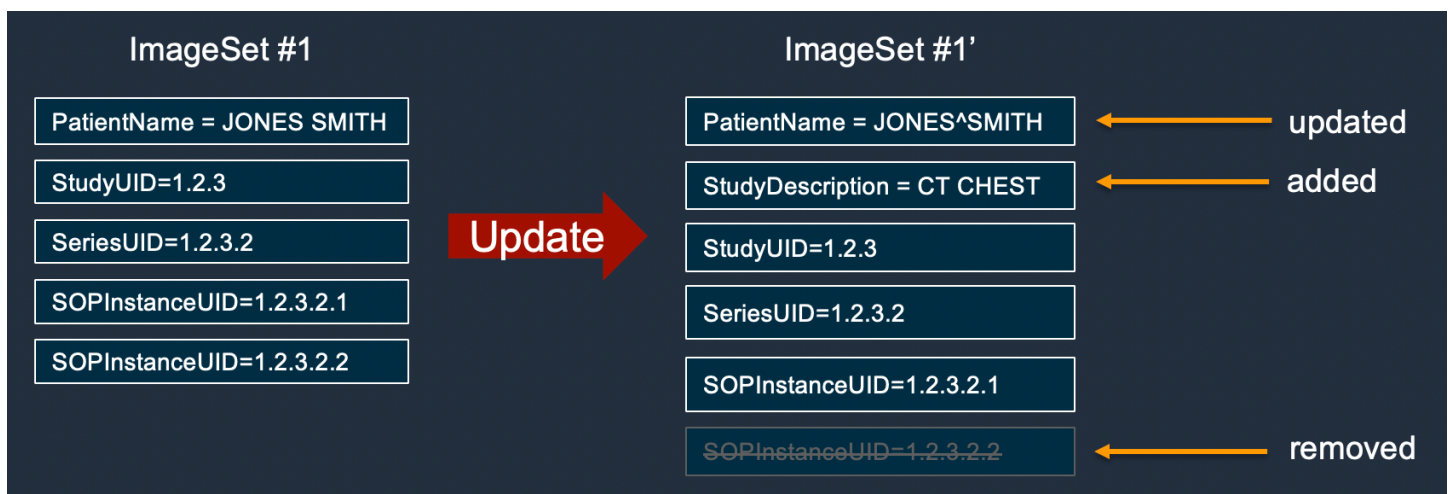
Note

Las importaciones DICOM reales requieren actualizar, añadir y eliminar atributos de los metadatos de los conjuntos 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](#).
- Se aplican restricciones de los elementos DICOM a las actualizaciones de metadatos. El parámetro de `force` solicitud le permite actualizar elementos de [conjuntos de imágenes](#) no principales en los casos en que desee [Restricciones de los metadatos de DICOM](#) anularlos.
- Los elementos de metadatos a nivel de paciente y serie no se pueden actualizar para los conjuntos de [imágenes](#) principales. [No UpdateImageSet admitirá la actualización del StudyInstance UID, el UID y el SeriesInstance UID de los SOPInstance conjuntos de imágenes principales. force](#)

- [Establezca el parámetro de forcesolicitud para forzar la finalización de la UpdateImageSetMetadata acción en los conjuntos de imágenes no principales.](#) La configuración de este parámetro permite realizar 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 datos DICOM privados a nivel de instancia
 - La acción de convertir un conjunto de imágenes en principal cambiará el ID del conjunto de imágenes.

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

El siguiente ejemplo de `update-image-set-metadata` 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: eliminar un atributo de los metadatos del conjunto de imágenes

El siguiente ejemplo de update-image-set-metadata 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: eliminar una instancia de los metadatos del conjunto de imágenes

El siguiente ejemplo de `update-image-set-metadata` 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 \
  --force
```

Contenido de `metadata-updates.json`

```
{
  "DICOMUpdates": {
    "removableAttributes": "{\"SchemaVersion\": 1.1,\"Study\":{\"Series\": {\"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}}"
  }
}
```

Salida:

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

Ejemplo 4: 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: añadir un elemento de datos DICOM privado a una instancia

El siguiente ejemplo de `update-image-set-metadata` muestra cómo añadir un elemento privado a una instancia específica en un conjunto de imágenes. El estándar DICOM permite

que los elementos de datos privados comuniquen información que no puede estar contenida en elementos de datos estándar. Puede crear, actualizar y eliminar elementos de datos privados con la `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: actualizar un elemento de datos DICOM privado a una instancia

El siguiente ejemplo de `update-image-set-metadata` muestra cómo actualizar el valor de un elemento de datos privado que pertenece a una instancia en 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
\\\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances
\\\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":
{\\\"00091001\\\": \\\"GE_GENESIS_DD\\\"}}}}}}}"
  }
}
```

Salida:

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

Ejemplo 7: Para actualizar un SOPInstance UID con el parámetro force

El siguiente update-image-set-metadata ejemplo muestra cómo actualizar un SOPInstance UID mediante el parámetro force para anular las restricciones de los metadatos DICOM.

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

```
--update-image-set-metadata-updates file://metadata-updates.json
```

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 la AWS HealthImaging Guía para desarrolladores.

- Para obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
/**
 * Update the metadata of an AWS HealthImaging image set.
 */
```

```

    * @param medicalImagingClient - The AWS HealthImaging client object.
    * @param datastoreId           - The datastore ID.
    * @param imageSetId           - The image set ID.
    * @param versionId            - The version ID.
    * @param metadataUpdates      - A MetadataUpdates object containing the
updates.
    * @param force                 - The force flag.
    * @throws MedicalImagingException - Base exception for all service
exceptions thrown by AWS HealthImaging.
    */
    public static void updateMedicalImageSetMetadata(MedicalImagingClient
medicalImagingClient,
                                                    String datastoreId,
                                                    String imageSetId,
                                                    String versionId,
                                                    MetadataUpdates
metadataUpdates,
                                                    boolean force) {
        try {
            UpdateImageSetMetadataRequest updateImageSetMetadataRequest =
UpdateImageSetMetadataRequest
                .builder()
                .datastoreId(datastoreId)
                .imageSetId(imageSetId)
                .latestVersionId(versionId)
                .updateImageSetMetadataUpdates(metadataUpdates)
                .force(force)
                .build();

            UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

```

Caso de uso 1: 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 n.º 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 n.º 3: eliminar una instancia.

```

        final String removeInstance = ""
        {
            "SchemaVersion": 1.1,
            "Study": {
                "Series": {

"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                    "Instances": {

"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                        }
                    }
                }
            }
        }
        """;

        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: Revertir 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 obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 * version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
    datastoreId = "xxxxxxxxxx",
    imageSetId = "xxxxxxxxxx",
    latestVersionId = "1",
    updateMetadata = "{}",
    force = false,
) => {
    try {
        const response = await medicalImagingClient.send(
            new UpdateImageSetMetadataCommand({

```

```

        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
    }},
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 200,
    //     requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //     extendedRequestId: undefined,
    //     cfId: undefined,
    //     attempts: 1,
    //     totalRetryDelay: 0
    //   },
    //   createdAt: 2023-09-22T14:49:26.427Z,
    //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   imageSetState: 'LOCKED',
    //   imageSetWorkflowStatus: 'UPDATING',
    //   latestVersionId: '4',
    //   updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};

```

Caso de uso 1: Insertar o actualizar un atributo y forzar 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: Revertir a una versión anterior.

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

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

- Para obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param metadata: The image set metadata as a dictionary.
            For example {"DICOMUpdates": {"updatableAttributes":
                {"\"SchemaVersion\":1.1,\"Patient\":{\"\"DICOM\":{\"\"PatientName\":
                \"Garcia^Gloria\"}}}}"}
        :param force: Force the update.
        :return: The updated image set metadata.
        """
        try:
            updated_metadata =
self.health_imaging_client.update_image_set_metadata(
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                latestVersionId=version_id,
                updateImageSetMetadataUpdates=metadata,
                force=force,
```

```

    )
except ClientError as err:
    logger.error(
        "Couldn't update image set metadata. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return updated_metadata

```

El siguiente código crea una instancia del `MedicalImagingWrapper` objeto.

```

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

```

Caso de uso 1: Insertar o actualizar 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 n.º 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": {

"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
            "Instances": {

"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}

            }
        }
    }
}"""
metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

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

```

Caso de uso 4: Revertir a una versión anterior.

```

metadata = {"revertToVersionId": "1"}

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

```

- Para obtener más información sobre la API, consulta [UpdateImageSetMetadata](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    " iv_latest_version_id = '1'
    " iv_force = abap_false
    oo_result = lo_mig->updateimagesetmetadata(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id
        iv_latestversionid = iv_latest_version_id
        io_updateimagesetmetupdates = io_metadata_updates
        iv_force = iv_force ).
    DATA(lv_new_version) = oo_result->get_latestversionid( ).
    MESSAGE |Image set metadata updated to version: { lv_new_version }.| TYPE
'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
    MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
```

```
CATCH /aws1/cx_migvalidationex.  
MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Puede mover instancias de SOP entre conjuntos de imágenes, resolver conflictos entre elementos de metadatos y añadir o eliminar instancias de los conjuntos de imágenes principales mediante las teclas `CopyImageSet`, `UpdateImageSetMetadata` y `DeleteImageSet` APIs

Puede eliminar un conjunto de imágenes de la colección principal con esta `DeleteImageSet` acción.

Para actualizar los metadatos de un conjunto de imágenes principal

1. Utilice la `CopyImageSet` acción para crear un conjunto de imágenes no principal que sea una copia del conjunto de imágenes principal que desee modificar. Supongamos que se devuelve `103785414bc2c89330f7ce51bbd13f7a` como el ID del conjunto de imágenes no principal.

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

2. Utilice la `UpdateImageSetMetadata` acción para realizar cambios en el conjunto de imágenes no principal. (103785414bc2c89330f7ce51bbd13f7a) Por ejemplo, cambiar el ID del paciente.

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

3. Elimine el conjunto de imágenes principal que está modificando.

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

4. Utilice la `CopyImageSet` acción con el argumento `--promoteToPrimary` para añadir el conjunto de imágenes actualizado a la colección principal.

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

5. Elimine el conjunto de imágenes no principal.

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

Para hacer que un conjunto de imágenes no principal sea principal

1. Utilice la `UpdateImageSetMetadata` acción para resolver conflictos con los conjuntos de imágenes principales existentes.

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

2. Cuando se resuelvan los conflictos, utilice la `CopyImageSet` acción con el argumento `--promoteToPrimary` para añadir el conjunto de imágenes a la colección de conjuntos de imágenes principal.

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

3. Tras confirmar que la `CopyImageSet` acción se ha realizado correctamente, elimine el conjunto de imágenes no principal de origen.

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

Copia de conjuntos de imágenes

Utilice 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 referencia de la HealthImaging API de AWS.

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 (`imageSetState` `imageSetWorkflowStatus`) 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 instancia SOP UUIDs sea única dentro de un conjunto de imágenes.
- Puede copiar subconjuntos de instancias SOP utilizando. [copiableAttributes](#) Esto le permite elegir una o más instancias de SOP de las que desea copiar `sourceImageSet` a `destinationImageSet`
- Si la CopyImageSet acción no se realiza correctamente, llame a GetImageSet la `message` propiedad para revisarla. Para obtener más información, consulte [Obtención de las propiedades del conjunto de imágenes](#).
- Las importaciones DICOM reales pueden dar como resultado varios conjuntos de imágenes por serie DICOM. La CopyImageSet acción requiere `sourceImageSet` y debe `destinationImageSet` tener metadatos consistentes, 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 ejemplo de `copy-image-set` crea una copia duplicada de un conjunto de imágenes sin un 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 ejemplo de `copy-image-set` crea 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: copiar un subconjunto de instancias de un conjunto de imágenes de origen a un conjunto de imágenes de destino.

El siguiente ejemplo de `copy-image-set` copia una instancia de DICOM del conjunto de imágenes de origen al conjunto de imágenes de destino. El parámetro `force` se proporciona para anular las incoherencias en los atributos de nivel de 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 la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [CopyImageSet](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
/**
 * Copy an AWS HealthImaging image set.
```

```

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

                                           String datastoreId,
                                           String imageSetId,
                                           String latestVersionId,
                                           String destinationImageSetId,
                                           String destinationVersionId,
                                           boolean force,
                                           Vector<String> subsets) {

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

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

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

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

```

```

        // Optionally designate a destination image set.
        if (destinationImageSetId != null) {
            copyImageSetBuilder =
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()
                .imageSetId(destinationImageSetId)
                .latestVersionId(destinationVersionId)
                .build());
        }

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

        CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

```

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 obtener más información sobre la API, consulte [CopyImageSet](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

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

```

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

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

  params.copyImageSetInformation.dicomCopies = copySubsetsJson;
}

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

```

```

    //      },
    //      sourceImageSetProperties: {
    //          createdAt: 2023-09-22T14:49:26.427Z,
    //          imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
    //          imageSetId: 'xxxxxxxxxxxxxxxxx',
    //          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,
);

```

Copiar un subconjunto de un conjunto de imágenes con un destino y forzar la copia.

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

- Para obtener más información sobre la API, consulte [CopyImageSet](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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,
    )

```

Copiar 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 siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [CopyImageSet](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_source_image_set_id = '1234567890123456789012345678901234567890'
    " iv_source_version_id = '1'
    " iv_destination_image_set_id =
'1234567890123456789012345678901234567890' (optional)
    " iv_destination_version_id = '1' (optional)
    " iv_force = abap_false
    DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
        iv_latestversionid = iv_source_version_id ).
    DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(
        io_sourceimageset = lo_source_info ).
    IF iv_destination_image_set_id IS NOT INITIAL AND
        iv_destination_version_id IS NOT INITIAL.
        DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
            iv_imagesetid = iv_destination_image_set_id
            iv_latestversionid = iv_destination_version_id ).
        lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
            io_sourceimageset = lo_source_info
            io_destinationimageset = lo_dest_info ).
    ENDIF.
    oo_result = lo_mig->copyimageset(
        iv_datastoreid = iv_datastore_id
        iv_sourceimagesetid = iv_source_image_set_id
        io_copyimagesetinformatoin = lo_copy_info
        iv_force = iv_force ).
    DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
    DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
    MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
```

```
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
    MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Image set not found.' TYPE 'I'.  
CATCH /aws1/cx_migservicequotaexcdex.  
    MESSAGE 'Service quota exceeded.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [CopyImageSet](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Eliminación de un conjunto de imágenes

Utilice la `DeleteImageSet` acción para eliminar un [conjunto de imágenes](#) HealthImaging. Los menús siguientes proporcionan un procedimiento para el Consola de administración de AWS y ejemplos de código para el AWS CLI y AWS SDKs. Para obtener más información, consulte [DeleteImageSet](#) la referencia de la HealthImaging API de AWS.

Cómo eliminar 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. 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++

SDK para C++

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

```

    else {
        std::cerr << "Error deleting image set " << imageSetID << " from data
store "
                << datastoreID << ": " <<
                outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

```

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la referencia AWS SDK para C++ de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta [Eliminar un conjunto de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
    String datastoreId,
    String imagesetId) {
    try {
        DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
            .datastoreId(datastoreId)
            .imageSetId(imagesetId)
            .build();

        medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

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

- Para obtener más información sobre la API, consulta [DeleteImageSet](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->deleteimageset(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id ).
    MESSAGE 'Image set deleted.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

 Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace [Enviar comentarios](#) en la barra lateral derecha de esta página.

Etiquetado de 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 identificación personal (PII), información médica protegida (PHI) ni otra información confidencial en las etiquetas. Las etiquetas no se han diseñado para usarse con información privada o confidencial.

En los temas siguientes se describe cómo utilizar las operaciones de HealthImaging etiquetado mediante las letras Consola de administración de AWS 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

Utilice la [TagResource](#) acción para etiquetar [almacenes de datos](#) y [conjuntos de imágenes](#) en AWS HealthImaging. En los siguientes ejemplos de código se describe cómo utilizar la TagResource acción con Consola de administración de AWS AWS CLI, y AWS SDKs. Para obtener más información, consulte [Etiquetar AWS los recursos](#) en la 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. Seleccione Save changes (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 los recursos AWS HealthImaging](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [TagResource](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

        medicalImagingClient.tagResource(tagResourceRequest);

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

- Para obtener más información sobre la API, consulte [TagResource](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

  return response;
};
```

- Para obtener más información sobre la API, consulte [TagResource](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [TagResource](#) la AWS Referencia de API de SDK for Python (Boto3).

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.  
    " iv_resource_arn = 'arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012'  
    lo_mig->tagresource(  
        iv_resourcearn = iv_resource_arn  
        it_tags = it_tags ).  
    MESSAGE 'Resource tagged successfully.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Resource not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [TagResource](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace [Enviar comentarios](#) en la barra lateral derecha de esta página.

Listado de etiquetas de un recurso

Utilice la [ListTagsForResource](#) acción para enumerar las etiquetas de [los almacenes de datos](#) y [conjuntos de imágenes](#) en AWS HealthImaging. En los siguientes ejemplos de código se describe cómo utilizar la ListTagsForResource acción con Consola de administración de AWS, AWS CLI, y AWS SDKs. Para obtener más información, consulte [Etiquetar AWS los recursos](#) en la 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 los recursos AWS HealthImaging](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static ListTagsForResourceResponse
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
```

```

        String resourceArn) {
    try {
        ListTagsForResourceRequest listTagsForResourceRequest =
ListTagsForResourceRequest.builder()
            .resourceArn(resourceArn)
            .build();

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

    return null;
}

```

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 * store or image set.
 */
export const listTagsForResource = async (
    resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi",
) => {

```

```

const response = await medicalImagingClient.send(
  new ListTagsForResourceCommand({ resourceArn: resourceArn }),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   tags: { Deployment: 'Development' }
// }

return response;
};

```

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

```

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

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

```

```

:param resource_arn: The ARN of the resource.
:return: The list of tags.
"""
try:
    tags = self.health_imaging_client.list_tags_for_resource(
        resourceArn=resource_arn
    )
except ClientError as err:
    logger.error(
        "Couldn't list tags for resource. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return tags["tags"]

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [ListTagsForResource](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_resource_arn = 'arn:aws:medical-imaging:us-
    east-1:123456789012:datastore/12345678901234567890123456789012'

```

```

        oo_result = lo_mig->listtagsforresource( iv_resourcearn =
iv_resource_arn ).
        DATA(lt_tags) = oo_result->get_tags( ).
        DATA(lv_count) = lines( lt_tags ).
        MESSAGE |Found { lv_count } tags for resource.| TYPE 'I'.
    CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
    CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
    CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
    CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
    CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.

```

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace Enviar comentarios en la barra lateral derecha de esta página.

Eliminación de las etiquetas de un recurso

Utilice la [UntagResource](#) acción para desetiquetar [los almacenes de datos](#) y los [conjuntos de imágenes](#) en AWS HealthImaging. En los siguientes ejemplos de código se describe cómo utilizar la UntagResource acción con Consola de administración de AWS, AWS CLI, y AWS SDKs. Para obtener más información, consulte [Etiquetar AWS los recursos](#) en la Referencia general de AWS Guía.

Para retirar 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. Seleccione Save changes (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 los recursos AWS HealthImaging](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [UntagResource](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static void untagMedicalImagingResource(MedicalImagingClient  
medicalImagingClient,  
        String resourceArn,  
        Collection<String> tagKeys) {  
    try {  
        UntagResourceRequest untagResourceRequest =  
        UntagResourceRequest.builder()  
            .resourceArn(resourceArn)  
            .tagKeys(tagKeys)  
            .build();  
  
        medicalImagingClient.untagResource(untagResourceRequest);  
  
        System.out.println("Tags have been removed from the resource.");  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Para obtener más información sobre la API, consulte [UntagResource](#) la Referencia AWS SDK for Java 2.x de la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

  return response;
};
```

- Para obtener más información sobre la API, consulte [UntagResource](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

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

- Para obtener más información sobre la API, consulta [UntagResource](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
    lo_mig->untagresource(
        iv_resourcearn = iv_resource_arn
        it_tagkeys = it_tag_keys ).
    MESSAGE 'Resource untagged successfully.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [UntagResource](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

 Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicita un ejemplo de código mediante el enlace [Enviar comentarios](#) en la barra lateral derecha de esta página.

Ejemplos de código para HealthImaging usar AWS SDKs

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

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

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica a través de llamadas a varias funciones dentro del servicio o combinado con otros Servicios de AWS.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Ejemplos de código

- [Ejemplos básicos de HealthImaging uso AWS SDKs](#)
 - [Hola HealthImaging](#)
 - [Acciones de HealthImaging uso AWS SDKs](#)
 - [Úselo CopyImageSet con un AWS SDK o CLI](#)
 - [Úselo CreateDatastore con un AWS SDK o CLI](#)
 - [Úselo DeleteDatastore con un AWS SDK o CLI](#)
 - [Úselo DeleteImageSet con un AWS SDK o CLI](#)
 - [Úselo GetDICOMImportJob con un AWS SDK o CLI](#)
 - [Úselo GetDatastore con un AWS SDK o CLI](#)
 - [Úselo GetImageFrame con un AWS SDK o CLI](#)
 - [Úselo GetImageSet con un AWS SDK o CLI](#)
 - [Úselo GetImageSetMetadata con un AWS SDK o CLI](#)
 - [Úselo ListDICOMImportJobs con un AWS SDK o CLI](#)
 - [Úselo ListDatastores con un AWS SDK o CLI](#)
 - [Úselo ListImageSetVersions con un AWS SDK o CLI](#)
 - [Úselo ListTagsForResource con un AWS SDK o CLI](#)
 - [Úselo SearchImageSets con un AWS SDK o CLI](#)

- [Úselo StartDICOMImportJob con un AWS SDK o CLI](#)
- [Úselo TagResource con un AWS SDK o CLI](#)
- [Úselo UntagResource con un AWS SDK o CLI](#)
- [Úselo UpdateImageSetMetadata con un AWS SDK o CLI](#)
- [Escenarios de HealthImaging uso AWS SDKs](#)
 - [Comience con los conjuntos HealthImaging de imágenes y los marcos de imágenes mediante un AWS SDK](#)
 - [Etiquetar un almacén HealthImaging de datos mediante un SDK AWS](#)
 - [Etiquetar un conjunto HealthImaging de imágenes mediante un SDK AWS](#)

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](#)
 - [Úselo CopyImageSet con un AWS SDK o CLI](#)
 - [Úselo CreateDatastore con un AWS SDK o CLI](#)
 - [Úselo DeleteDatastore con un AWS SDK o CLI](#)
 - [Úselo DeleteImageSet con un AWS SDK o CLI](#)
 - [Úselo GetDICOMImportJob con un AWS SDK o CLI](#)
 - [Úselo GetDatastore con un AWS SDK o CLI](#)
 - [Úselo GetImageFrame con un AWS SDK o CLI](#)
 - [Úselo GetImageSet con un AWS SDK o CLI](#)
 - [Úselo GetImageSetMetadata con un AWS SDK o CLI](#)
 - [Úselo ListDICOMImportJobs con un AWS SDK o CLI](#)
 - [Úselo ListDatastores con un AWS SDK o CLI](#)
 - [Úselo ListImageSetVersions con un AWS SDK o CLI](#)
 - [Úselo ListTagsForResource con un AWS SDK o CLI](#)
 - [Úselo SearchImageSets con un AWS SDK o CLI](#)

- [Úselo StartDICOMImportJob con un AWS SDK o CLI](#)
- [Úselo TagResource con un AWS SDK o CLI](#)
- [Úselo UntagResource con un AWS SDK o CLI](#)
- [Úselo UpdateImageSetMetadata con un AWS SDK o CLI](#)

Hola HealthImaging

En los siguientes ejemplos de código se muestra cómo empezar a utilizar HealthImaging.

C++

SDK para C++

Código para el CMake archivo CMake Lists.txt.

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

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

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

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

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

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

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

```

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

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

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

add_executable(${PROJECT_NAME}
    hello_health_imaging.cpp)

target_link_libraries(${PROJECT_NAME}
    ${AWSSDK_LINK_LIBRARIES})

```

Código 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 obtener más información sobre la API, consulte [ListDatastores](#) la Referencia AWS SDK para C++ de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

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

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

    const { datastoreSummaries } = await client.send(command);

```

```
console.log("Datastores: ");
console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
return datastoreSummaries;
};
```

- Para obtener más información sobre la API, consulte [ListDatastores](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

logger = logging.getLogger(__name__)

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

    :param medical_imaging_client: A Boto3 AWS HealthImaging Client object.
    """
    print("Hello, Amazon Health Imaging! Let's list some of your data stores:\n")
    try:
        paginator = medical_imaging_client.get_paginator("list_datastores")
        page_iterator = paginator.paginate()
        datastore_summaries = []
        for page in page_iterator:
            datastore_summaries.extend(page["datastoreSummaries"])
```

```
print("\tData Stores:")
for ds in datastore_summaries:
    print(f"\t\tDatastore: {ds['datastoreName']} ID {ds['datastoreId']}")
except ClientError as err:
    logger.error(
        "Couldn't list data stores. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise

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

- Para obtener más información sobre la API, consulta [ListDatastores](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus 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 fragmentos de código de programas más grandes que deben ejecutarse en su contexto. Puede ver las acciones en contexto en [Escenarios de HealthImaging uso AWS SDKs](#).

Los siguientes ejemplos incluyen solo las acciones que se utilizan con mayor frecuencia. Para ver una lista completa, consulte la [Referencia de la API de AWS HealthImaging](#).

Ejemplos

- [Úselo CopyImageSet con un AWS SDK o CLI](#)
- [Úselo CreateDatastore con un AWS SDK o CLI](#)
- [Úselo DeleteDatastore con un AWS SDK o CLI](#)
- [Úselo DeleteImageSet con un AWS SDK o CLI](#)
- [Úselo GetDICOMImportJob con un AWS SDK o CLI](#)
- [Úselo GetDatastore con un AWS SDK o CLI](#)
- [Úselo GetImageFrame con un AWS SDK o CLI](#)
- [Úselo GetImageSet con un AWS SDK o CLI](#)
- [Úselo GetImageSetMetadata con un AWS SDK o CLI](#)
- [Úselo ListDICOMImportJobs con un AWS SDK o CLI](#)
- [Úselo ListDatastores con un AWS SDK o CLI](#)
- [Úselo ListImageSetVersions con un AWS SDK o CLI](#)
- [Úselo ListTagsForResource con un AWS SDK o CLI](#)
- [Úselo SearchImageSets con un AWS SDK o CLI](#)
- [Úselo StartDICOMImportJob con un AWS SDK o CLI](#)
- [Úselo TagResource con un AWS SDK o CLI](#)
- [Úselo UntagResource con un AWS SDK o CLI](#)
- [Úselo UpdateImageSetMetadata con un AWS SDK o CLI](#)

Úselo **CopyImageSet** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar CopyImageSet.

CLI

AWS CLI

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

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

```
aws medical-imaging copy-image-set \  
  --datastore-id 12345678901234567890123456789012 \  
  --image-set-id 12345678901234567890123456789012 \  
  --tags key=value
```

```
--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 ejemplo de `copy-image-set` crea 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: copiar un subconjunto de instancias de un conjunto de imágenes de origen a un conjunto de imágenes de destino.

El siguiente ejemplo de `copy-image-set` copia una instancia de DICOM del conjunto de imágenes de origen al conjunto de imágenes de destino. El parámetro `force` se proporciona para anular las incoherencias en los atributos de nivel de 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 la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [CopyImageSet](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```

/**
 * Copy an AWS HealthImaging image set.
 *
 * @param medicalImagingClient - The AWS HealthImaging client object.
 * @param datastoreId          - The datastore ID.
 * @param imageSetId           - The image set ID.
 * @param latestVersionId      - The version ID.
 * @param destinationImageSetId - The optional destination image set ID,
 * ignored if null.
 * @param destinationVersionId - The optional destination version ID,
 * ignored if null.
 * @param force                 - The force flag.
 * @param subsets               - The optional subsets to copy, ignored if
 * null.
 * @return                      - The image set ID of the copy.

```

```

    * @throws MedicalImagingException - Base exception for all service
    exceptions thrown by AWS HealthImaging.
    */
    public static String copyMedicalImageSet(MedicalImagingClient
medicalImagingClient,

                                            String datastoreId,
                                            String imageSetId,
                                            String latestVersionId,
                                            String destinationImageSetId,
                                            String destinationVersionId,
                                            boolean force,
                                            Vector<String> subsets) {

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

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

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

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

            // Optionally designate a destination image set.
            if (destinationImageSetId != null) {
                copyImageSetBuilder =
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()
                    .imageSetId(destinationImageSetId)
                    .latestVersionId(destinationVersionId)
                    .build());
            }

            CopyImageSetRequest copyImageSetRequest =
CopyImageSetRequest.builder()
                .datastoreId(datastoreId)

```

```

        .sourceImageSetId(imageSetId)
        .copyImageSetInformation(copyImageSetBuilder.build())
        .force(force)
        .build();

    CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

```

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 obtener más información sobre la API, consulte [CopyImageSet](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

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

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

```

```

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

    params.copyImageSetInformation.dicomCopies = copySubsetsJson;
}

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

```

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

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,  
);
```

Copiar un subconjunto de un conjunto de imágenes con un destino y forzar la copia.

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

- Para obtener más información sobre la API, consulte [CopyImageSet](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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,
)
```

Copiar 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 siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [CopyImageSet](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_source_image_set_id = '1234567890123456789012345678901234567890'
    " iv_source_version_id = '1'
    " iv_destination_image_set_id =
'1234567890123456789012345678901234567890' (optional)
    " iv_destination_version_id = '1' (optional)
    " iv_force = abap_false
    DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
        iv_latestversionid = iv_source_version_id ).
    DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(
        io_sourceimageset = lo_source_info ).
    IF iv_destination_image_set_id IS NOT INITIAL AND
        iv_destination_version_id IS NOT INITIAL.
        DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
            iv_imagesetid = iv_destination_image_set_id
            iv_latestversionid = iv_destination_version_id ).
        lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
            io_sourceimageset = lo_source_info
            io_destinationimageset = lo_dest_info ).
    ENDIF.
    oo_result = lo_mig->copyimageset(
        iv_datastoreid = iv_datastore_id
        iv_sourceimagesetid = iv_source_image_set_id
        io_copyimagesetinformatio = lo_copy_info
        iv_force = iv_force ).
    DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
    DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
    MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
    MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.

```

```

    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.

```

- Para obtener más información sobre la API, consulte [CopyImageSet](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **CreateDatastore** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 getopt command in a function.

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

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

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

```

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

local error_code=${?}

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

echo "$response"

return 0
}
```

- Para obtener detalles sobre la API, consulte [CreateDatastore](#) la Referencia de AWS CLI comandos.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

CLI

AWS CLI

Ejemplo 1: 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. Al crear un almacén de datos sin especificar a--lossless-storage-format, el AWS HealthImaging valor predeterminado es HTJ2 K (JPEG 2000 de alto rendimiento).

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

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

Salida:

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

Ejemplo 2: Creación de un almacén de datos con el formato de almacenamiento JPEG 2000 sin pérdidas

Un almacén de datos configurado con el formato de almacenamiento JPEG 2000 sin pérdidas transcodificará y conservará los fotogramas de imagen sin pérdidas en formato JPEG 2000. A continuación, los fotogramas de imagen se pueden recuperar en JPEG 2000 sin pérdidas sin necesidad de transcodificación. En el ejemplo de código `create-datastore` siguiente, se crea un almacén de datos configurado para el formato de almacenamiento JPEG 2000 sin pérdidas con el nombre `my-datastore`.

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

Salida:

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

Para obtener más información, consulte [Creación de un banco de datos en la Guía para desarrolladores](#).AWS HealthImaging

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

    return "";
}
```

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

/**
 * @param {string} datastoreName - The name of the data store to create.
```

```

*/
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'CREATING'
  // }
  return response;
};

```

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

```

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

    def create_datastore(self, name):
        """

```

```

    Create a data store.

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

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [CreateDatastore](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_name = 'my-datastore-name'

```

```

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

```

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **DeleteDatastore** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 getopt command in a function.

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

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

```

```
    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 obtener detalles sobre la API, consulte [DeleteDatastore](#) la Referencia de AWS CLI comandos.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta [Eliminar un banco de datos](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript**SDK para JavaScript (v3)**

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

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

  return response;
};
```

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [DeleteDatastore](#) la AWS Referencia de API de SDK for Python (Boto3).

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.  
    " iv_datastore_id = '1234567890123456789012345678901234567890'  
    oo_result = lo_mig->deletedatastore( iv_datastoreid = iv_datastore_id ).  
    MESSAGE 'Data store deleted.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
    MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
    MESSAGE 'Conflict. Data store may contain resources.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
    MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
    MESSAGE 'Data store not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
    MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
    MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **DeleteImageSet** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 su contexto en el siguiente ejemplo de código:

- [Introducción a los conjuntos y marcos de imágenes](#)

C++

SDK para C++

```

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

```

```
        outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la referencia AWS SDK para C++ de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta [Eliminar un conjunto de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
    String datastoreId,
    String imagesetId) {
    try {
        DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
            .datastoreId(datastoreId)
            .imageSetId(imagesetId)
            .build();

        medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

/**
```

```

* @param {string} datastoreId - The data store ID.
* @param {string} imageSetId - The image set ID.
*/
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};

```

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [DeleteImageSet](#) la AWS Referencia de API de SDK for Python (Boto3).

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.  
  " iv_datastore_id = '1234567890123456789012345678901234567890'  
  " iv_image_set_id = '1234567890123456789012345678901234567890'  
  oo_result = lo_mig->deleteimageset(  
    iv_datastoreid = iv_datastore_id  
    iv_imagesetid = iv_image_set_id ).  
  MESSAGE 'Image set deleted.' TYPE 'I'.  
CATCH /aws1/cx_migaccessdeniedex.  
  MESSAGE 'Access denied.' TYPE 'I'.  
CATCH /aws1/cx_migconflictexception.  
  MESSAGE 'Conflict error.' TYPE 'I'.  
CATCH /aws1/cx_miginternalserverex.  
  MESSAGE 'Internal server error.' TYPE 'I'.  
CATCH /aws1/cx_migresourcenotfoundex.  
  MESSAGE 'Image set not found.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
  MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
  MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **GetDICOMImportJob** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 su contexto en el siguiente ejemplo de código:

- [Introducción a los conjuntos y marcos de imágenes](#)

C++

SDK para C++

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

    return outcome;
}
```

```
}
```

- Para obtener información sobre la API, consulta [Get DICOMImport Job](#) in AWS SDK para C++ API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta [Cómo obtener propiedades de trabajos de importación](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener información sobre la API, consulte [Get DICOMImport Job](#) in AWS CLI Command Reference.

Java

SDK para Java 2.x

```
public static DICOMImportJobProperties getDicomImportJob(MedicalImagingClient  
medicalImagingClient,  
                String datastoreId,  
                String jobId) {  
  
    try {  
        GetDicomImportJobRequest getDicomImportJobRequest =  
        GetDicomImportJobRequest.builder()  
            .datastoreId(datastoreId)  
            .jobId(jobId)  
            .build();  
        GetDicomImportJobResponse response =  
        medicalImagingClient.getDICOMImportJob(getDicomImportJobRequest);  
        return response.jobProperties();  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
  
    return null;  
}
```

- Para obtener información sobre la API, consulta [Get DICOMImport Job](#) in AWS SDK for Java 2.x API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript**SDK para JavaScript (v3)**

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxx",
) => {
  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: 'xxxxxxxxxxxxxxxxxxxxxx',
  //     endedAt: 2023-09-19T17:29:21.753Z,
  //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //     jobName: 'job_1',
  //     jobStatus: 'COMPLETED',
  //   },
  // }
```

```

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

    return response;
};

```

- Para obtener información sobre la API, consulta [Get DICOMImport Job](#) in AWS SDK para JavaScript API Reference.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param job_id: The ID of the job.
        :return: The job properties.
        """
        try:
            job = self.health_imaging_client.get_dicom_import_job(
                jobId=job_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
```

```

        "Couldn't get DICOM import job. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return job["jobProperties"]

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para [obtener más información sobre la API, consulta la referencia de la API Get DICOMImport Job](#) in AWS SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_job_id = '12345678901234567890123456789012'
    oo_result = lo_mig->getdicomimportjob(
        iv_datastoreid = iv_datastore_id
        iv_jobid = iv_job_id ).
    DATA(lo_job_props) = oo_result->get_jobproperties( ).
    DATA(lv_job_status) = lo_job_props->get_jobstatus( ).
    MESSAGE |Job status: { lv_job_status }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.

```

```

CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Job not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.

```

- Para [obtener más información sobre la API, consulte la referencia Get DICOMImport Job](#) in AWS SDK para la API ABAP de SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **GetDatastore** con un AWS SDK o CLI

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

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

    if [[ -z "$datastore_id" ]]; then

```

```
errecho "ERROR: You must provide a data store ID with the -i parameter."
usage
return 1
fi

local response

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

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

echo "$response"

return 0
}
```

- Para obtener detalles sobre la API, consulte [GetDatastore](#) la Referencia de AWS CLI comandos.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

CLI

AWS CLI

Ejemplo 1: 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",  
    "losslessStorageFormat": "HTJ2K"  
    "datastoreArn": "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/12345678901234567890123456789012",  
    "createdAt": "2022-11-15T23:33:09.643000+00:00",  
    "updatedAt": "2022-11-15T23:33:09.643000+00:00"  
  }  
}
```

Ejemplo 2: Para configurar las propiedades del almacén de datos para JPEG2 000

En el ejemplo de código `get-datastore` siguiente, se obtienen las propiedades de un almacén de datos configurado para el formato de almacenamiento JPEG 2000 sin pérdidas.

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

Salida:

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

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

Para obtener más información, consulte [Obtener las propiedades del almacén de datos](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetDatastore](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

    return null;
}
```

- Para obtener más información sobre la API, consulte [GetDatastore](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript**SDK para JavaScript (v3)**

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

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

- Para obtener más información sobre la API, consulte [GetDatastore](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [GetDatastore](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->getdatastore( iv_datastoreid = iv_datastore_id ).
    DATA(lo_properties) = oo_result->get_datastoreproperties( ).
    DATA(lv_name) = lo_properties->get_datastorename( ).
    DATA(lv_status) = lo_properties->get_datastorestatus( ).
    MESSAGE 'Data store properties retrieved.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Data store not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [GetDatastore](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **GetImageFrame** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 su contexto en el siguiente ejemplo de código:

- [Introducción a los conjuntos y marcos de imágenes](#)

C++

SDK para C++

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

```
Aws::MedicalImaging::MedicalImagingClient client(clientConfig);

Aws::MedicalImaging::Model::GetImageFrameRequest request;
request.SetDatastoreId(dataStoreID);
request.SetImageSetId(imageSetID);

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

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

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

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

}

return outcome.IsSuccess();
}
```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la referencia AWS SDK para C++ de la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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.jpg
```

Nota: Este ejemplo de código no incluye la salida porque la `GetImageFrame` acción devuelve un flujo de datos de píxeles al archivo `imageframe.jpg`. Para obtener información sobre la decodificación y la visualización de marcos de imágenes, consulte las bibliotecas de decodificación K. HTJ2

Para obtener más información, consulte [Obtener datos de píxeles de conjuntos de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

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

```

        .imageFrameInformation(ImageFrameInformation.builder()

        .imageFrameId(imageFrameId)

        .build())

        .build();

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreId - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.

```

```

*/
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

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

```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [GetImageFrame](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    " iv_image_frame_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->getimageframe(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id
        io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(
            iv_imageframeid = iv_image_frame_id ) ).
    DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).
    MESSAGE 'Image frame retrieved.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image frame not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [GetImageFrame](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **GetImageSet** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetImageSet](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static GetImageSetResponse getMedicalImageSet(MedicalImagingClient  
medicalImagingClient,  
            String datastoreId,  
            String imagesetId,  
            String versionId) {  
    try {  
        GetImageSetRequest.Builder getImageSetRequestBuilder =  
        GetImageSetRequest.builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imagesetId);  
  
        if (versionId != null) {  
            getImageSetRequestBuilder =  
            getImageSetRequestBuilder.versionId(versionId);  
        }  
  
        return  
        medicalImagingClient.getImageSet(getImageSetRequestBuilder.build());  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
  
    return null;  
}
```

- Para obtener más información sobre la API, consulte [GetImageSet](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript**SDK para JavaScript (v3)**

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

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

```
//      datastoreId: 'xxxxxxxxxxxxxxxxxx',  
//      imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/  
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxxxxxxx',  
//      imageSetId: 'xxxxxxxxxxxxxxxxxx',  
//      imageSetState: 'ACTIVE',  
//      imageSetWorkflowStatus: 'CREATED',  
//      updatedAt: 2023-09-22T14:49:26.427Z,  
//      versionId: '1'  
// }  
  
    return response;  
};
```

- Para obtener más información sobre la API, consulte [GetImageSet](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
```

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

El siguiente código crea una instancia del `MedicalImagingWrapper` objeto.

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

- Para obtener más información sobre la API, consulta [GetImageSet](#) la AWS Referencia de API de SDK for Python (Boto3).

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

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

- Para obtener más información sobre la API, consulte [GetImageSet](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **GetImageSetMetadata** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 su contexto en el siguiente ejemplo de código:

- [Introducción a los conjuntos y marcos de imágenes](#)

C++

SDK para 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 obtener más información sobre la API, consulte [GetImageSetMetadata](#) la referencia AWS SDK para C++ de la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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 objeto JSON 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 objeto JSON devuelto, primero debe descomprimirlo.

Salida:

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

Para obtener más información, consulta [Cómo obtener metadatos de conjuntos de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [GetImageSetMetadata](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static void getMedicalImageSetMetadata(MedicalImagingClient  
medicalImagingClient,  
    String destinationPath,  
    String datastoreId,  
    String imagesetId,  
    String versionId) {  
  
    try {  
        GetImageSetMetadataRequest.Builder getImageSetMetadataRequestBuilder  
= GetImageSetMetadataRequest.builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imagesetId);  
  
        if (versionId != null) {  
            getImageSetMetadataRequestBuilder =  
getImageSetMetadataRequestBuilder.versionId(versionId);  
        }  
    }  
}
```

```

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

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

```

- Para obtener más información sobre la API, consulte [GetImageSetMetadata](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para 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 "node:fs";

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

```

```

    imageSetId = "xxxxxxxxxxxxxx",
    versionID = "",
  ) => {
    const params = { datastoreId: datastoreId, imageSetId: imageSetId };

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

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

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

    return response;
  };

```

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 obtener más información sobre la API, consulte [GetImageSetMetadata](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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 MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [GetImageSetMetadata](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    " iv_version_id = '1' (optional)
    IF iv_version_id IS NOT INITIAL.

```

```

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

```

- Para obtener más información sobre la API, consulte [GetImageSetMetadata](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **ListDICOMImportJobs** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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, consulta Cómo [enumerar los trabajos de importación](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en la referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

```

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

    return new ArrayList<>();
}

```

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en la referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

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

```

```

    pageSize: 50,
  };

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

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

  return jobSummaries;
};

```

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en la referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta la [sección Lista de DICOMImport trabajos](#) en la referencia de la API del AWS SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

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

- Para obtener más información sobre la API, consulte [Listar DICOMImport trabajos](#) en el AWS SDK para obtener una referencia sobre la API ABAP de SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **ListDatastores** con un AWS SDK o CLI

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

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

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

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

echo "$response"

return 0
}
```

- Para obtener detalles sobre la API, consulte [ListDatastores](#) la Referencia de AWS CLI comandos.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta la sección sobre cómo [enumerar los almacenes de datos](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [ListDatastores](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

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

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

    return null;
}
```

- Para obtener más información sobre la API, consulte [ListDatastores](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

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

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

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

  return datastoreSummaries;
}
```

```
};
```

- Para obtener más información sobre la API, consulte [ListDatastores](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

```
return datastore_summaries
```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [ListDatastores](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

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

- Para obtener más información sobre la API, consulte [ListDatastores](#) la referencia sobre la API ABAP del AWS SDK para SAP.

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **ListImageSetVersions** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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": 1680027126.436  
    }  
  ]  
}
```

```

        "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, consulta la sección sobre [la lista de versiones de conjuntos de imágenes](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```

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

```

```

        .imageSetId(imagesetId)
        .build();

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

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

return null;
}

```

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
    datastoreId = "xxxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxx",

```

```

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

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

  const imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    imageSetPropertiesList.push(...page.imageSetPropertiesList);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   imageSetPropertiesList: [
  //     {
  //       ImageSetWorkflowStatus: 'CREATED',
  //       createdAt: 2023-09-22T14:49:26.427Z,
  //       imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //       imageSetState: 'ACTIVE',
  //       versionId: '1'
  //     }
  //   ]
  // }
  return imageSetPropertiesList;
};

```

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la Referencia de AWS SDK para JavaScript la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :return: The list of image set versions.
        """
        try:
            paginator = self.health_imaging_client.get_paginator(
                "list_image_set_versions"
            )
            page_iterator = paginator.paginate(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
            image_set_properties_list = []
            for page in page_iterator:
                image_set_properties_list.extend(page["imageSetPropertiesList"])
        except ClientError as err:
            logger.error(
                "Couldn't list image set versions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return image_set_properties_list
```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [ListImageSetVersions](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_image_set_id = '1234567890123456789012345678901234567890'
    oo_result = lo_mig->listimagesetversions(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id ).
    DATA(lt_versions) = oo_result->get_imagesetpropertieslist( ).
    DATA(lv_count) = lines( lt_versions ).
    MESSAGE |Found { lv_count } image set versions.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
```

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

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **ListTagsForResource** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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, consulta Cómo [etiquetar los recursos AWS HealthImaging](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static ListTagsForResourceResponse
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
    String resourceArn) {
    try {
```

```

        ListTagsForResourceRequest listTagsForResourceRequest =
ListTagsForResourceRequest.builder()
    .resourceArn(resourceArn)
    .build();

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

    return null;
}

```

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

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

```

```

    );
    console.log(response);
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //     tags: { Deployment: 'Development' }
    // }

    return response;
};

```

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

```

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

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

        :param resource_arn: The ARN of the resource.
        :return: The list of tags.

```

```

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

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [ListTagsForResource](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
    oo_result = lo_mig->listtagsforresource( iv_resourcearn =
iv_resource_arn ).

```

```
DATA(lt_tags) = oo_result->get_tags( ).
DATA(lv_count) = lines( lt_tags ).
MESSAGE |Found { lv_count } tags for resource.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [ListTagsForResource](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **SearchImageSets** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 su contexto en el siguiente ejemplo de código:

- [Introducción a los conjuntos y marcos de imágenes](#)

C++

SDK para 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();
        }
    } while (result);
}

```

```

    }
    else {
        std::cout << "Error: " << outcome.GetError().GetMessage() <<
std::endl;
        result = false;
    }
} while (!nextToken.empty());

return result;
}

```

Caso de uso núm. 1: operador IGUAL.

```

    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::Operator::EQUALS),
    .WithValues({Aws::MedicalImaging::Model::SearchByAttributeValue().WithDICOMPatientId(patientID)});

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

    searchCriteriaEqualsPatientID,

    imageIDsForPatientID,

                                                                    clientConfig);

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

```

Caso de uso #2: el operador BETWEEN usa DICOMStudy fecha y DICOMStudy hora.

```

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

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

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

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

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

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

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

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

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

```

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```

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

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

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

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

```

```

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

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

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

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

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

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

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

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

        useCase4SearchCriteria.SetSort(useCase4Sort);

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

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

```

- Para obtener más información sobre la API, consulta la Referencia de la API.
[SearchImageSets](#) AWS SDK para C++

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

CLI

AWS CLI

Ejemplo 1: búsqueda de conjuntos de imágenes con un operador EQUAL

En el siguiente ejemplo de código `search-image-sets` se usa el operador EQUAL 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 operador BETWEEN mediante DICOMStudy fecha y DICOMStudy hora

En el siguiente ejemplo de código `search-image-sets` se buscan conjuntos de imágenes con estudios DICOM generados entre el 1 de enero de 1990 (00:00 h) y el 1 de enero de 2023 (00:00 h).

Nota: La DICOMStudy hora 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: búsqueda de conjuntos de imágenes con un operador BETWEEN mediante createdAt (los estudios de tiempo se conservaban previamente)

El siguiente ejemplo de search-image-sets código busca conjuntos de imágenes cuyos estudios DICOM persistan HealthImaging entre los intervalos de tiempo de la zona horaria UTC.

Nota: Ingrese createdAt en el formato de ejemplo ("1985-04-12T23:20:50.52Z").

```

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 operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordenar la respuesta en orden ASC en el campo UpdatedAt

El siguiente ejemplo de `search-image-sets` código busca conjuntos de imágenes con un operador `EQUAL` en `DICOMSeries InstanceUID` y `BETWEEN` en `UpdatedAt` y ordena la respuesta en orden `ASC` en el campo `UpdatedAt`.

Nota: Introduzca `updatedAt` en el formato de ejemplo ("1985-04-12T23:20:50.52Z").

```
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, consulta la sección [Búsqueda de conjuntos de imágenes en la Guía para desarrolladores.AWS HealthImaging](#)

- Para obtener más información sobre la API, consulte [SearchImageSets](#) la Referencia de AWS CLI comandos.

Java

SDK para 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 núm. 1: operador IGUAL.

```

        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: ENTRE el operador que usa DICOMStudy fecha y DICOMStudy hora.

```

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMdd");
        searchFilters = Collections.singletonList(SearchFilter.builder()
            .operator(Operator.BETWEEN)
            .values(SearchByAttributeValue.builder()

```

```

        .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                                .dicomStudyDate("19990101")
                                .dicomStudyTime("000000.000")
                                .build())
        .build(),
        SearchByAttributeValue.builder()

        .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                                .dicomStudyDate(LocalDate.now()
                                                .format(formatter))
                                .dicomStudyTime("000000.000")
                                .build())
        .build())
        .build());

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

imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
    datastoreId, searchCriteria);
if (imageSetsMetadataSummaries != null) {
    System.out.println(
        "The image sets searched with BETWEEN operator using
DICOMStudyDate and DICOMStudyTime are:\n"
        +
        imageSetsMetadataSummaries);
    System.out.println();
}

```

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```

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

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

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

```

```

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

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

```

- Para obtener más información sobre la API, consulta la Referencia de la API.

[SearchImageSets](#) AWS SDK for Java 2.x

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para 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 núm. 1: operador IGUAL.

```
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: el operador BETWEEN usa DICOMStudy fecha y DICOMStudy hora.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [
          {
            DICOMStudyDateAndTime: {
              DICOMStudyDate: "19900101",
              DICOMStudyTime: "000000",
            },
          },
          {
            DICOMStudyDateAndTime: {
              DICOMStudyDate: "20230901",
              DICOMStudyTime: "000000",
            },
          },
        ],
      },
    ],
  };

  await searchImageSets(datastoreId, searchCriteria);
}
```

```

        ],
        operator: "BETWEEN",
      },
    ],
  };

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

```

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```

const datastoreId = "12345678901234567890123456789012";

try {

```

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

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

- Para obtener más información sobre la API, consulta la Referencia de la API. [SearchImageSets](#) AWS SDK para JavaScript

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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 núm. 1: operador IGUAL.

```

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: el operador BETWEEN usa DICOMStudy fecha y DICOMStudy hora.

```

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

image_sets = self.search_image_sets(data_store_id, search_filter)
print(
    f"Image sets found with BETWEEN operator using DICOMStudyDate and\nDICOMStudyTime\n{image_sets}"
)

```

Caso de uso núm. 3: el operador ENTRE usa CreatedAt. 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: operador EQUAL en DICOMSeries InstanceUID y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```
search_filter = {
    "filters": [
        {
            "values": [
                {
                    "updatedAt": datetime.datetime(
                        2021, 8, 4, 14, 49, 54, 429000
                    )
                },
                {
                    "updatedAt": datetime.datetime.now()
                },
            ],
            "operator": "BETWEEN",
        }
    ]
}
```

```

        + datetime.timedelta(days=1)
    },
    ],
    "operator": "BETWEEN",
  },
  {
    "values": [{"DICOMSeriesInstanceUID": series_instance_uid}],
    "operator": "EQUAL",
  },
],
"sort": {
  "sortOrder": "ASC",
  "sortField": "updatedAt",
},
}

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

```

MedicalImagingWrapper El siguiente código crea una instancia del objeto.

```

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

```

- Para obtener más información sobre la API, consulta [SearchImageSets](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
  " iv_datastore_id = '1234567890123456789012345678901234567890'
  oo_result = lo_mig->searchimagesets(
    iv_datastoreid = iv_datastore_id
    io_searchcriteria = io_search_criteria ).
  DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).
  DATA(lv_count) = lines( lt_imagesets ).
  MESSAGE |Found { lv_count } image sets.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
  MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
  MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
  MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
  MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
  MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
  MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [SearchImageSets](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **StartDICOMImportJob** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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 su contexto en el siguiente ejemplo de código:

- [Introducción a los conjuntos y marcos de imágenes](#)

C++

SDK para C++

```

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

```

```

startDICOMImportJobRequest.SetOutputS3Uri(outputURI);

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

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

return startDICOMImportJobOutcome.IsSuccess();
}

```

- Para obtener más información sobre la API, consulta [Start DICOMImport Job](#) in AWS SDK para C++ API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

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, consulta Cómo [iniciar un trabajo de importación](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener información sobre la API, consulte [Start DICOMImport Job](#) in AWS CLI Command Reference.

Java

SDK para Java 2.x

```
public static String startDicomImportJob(MedicalImagingClient  
medicalImagingClient,  
    String jobName,  
    String datastoreId,  
    String dataAccessRoleArn,  
    String inputS3Uri,  
    String outputS3Uri) {  
  
    try {  
        StartDicomImportJobRequest startDicomImportJobRequest =  
        StartDicomImportJobRequest.builder()  
            .jobName(jobName)  
            .datastoreId(datastoreId)  
            .dataAccessRoleArn(dataAccessRoleArn)  
            .inputS3Uri(inputS3Uri)  
            .outputS3Uri(outputS3Uri)  
            .build();  
  
        StartDicomImportJobResponse response =  
        medicalImagingClient.startDICOMImportJob(startDicomImportJobRequest);
```

```

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

    return "";
}

```

- Para obtener más información sobre la API, consulta [Start DICOMImport Job](#) in AWS SDK for Java 2.x API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

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

```

```

) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   jobStatus: 'SUBMITTED',
  //   submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};

```

- Para obtener más información sobre la API, consulta [Start DICOMImport Job](#) in AWS SDK para JavaScript API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

        :param job_name: The name of the job.
        :param datastore_id: The ID of the data store.
        :param role_arn: The Amazon Resource Name (ARN) of the role to use for
the job.
        :param input_s3_uri: The S3 bucket input prefix path containing the DICOM
files.
        :param output_s3_uri: The S3 bucket output prefix path for the result.
        :return: The job ID.
        """
        try:
            job = self.health_imaging_client.start_dicom_import_job(
                jobName=job_name,
                datastoreId=datastore_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_s3_uri,
                outputS3Uri=output_s3_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta la referencia de la API [Start DICOMImport Job](#) in AWS SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '1234567890123456789012345678901234567890'
    " iv_role_arn = 'arn:aws:iam::123456789012:role/ImportJobRole'
    " iv_input_s3_uri = 's3://my-bucket/input/'
    " iv_output_s3_uri = 's3://my-bucket/output/'
    oo_result = lo_mig->startdicomimportjob(
        iv_jobname = iv_job_name
        iv_datastoreid = iv_datastore_id
        iv_dataaccessrolearn = iv_role_arn
        iv_inputs3uri = iv_input_s3_uri
        iv_outputs3uri = iv_output_s3_uri ).
    DATA(lv_job_id) = oo_result->get_jobid( ).
    MESSAGE |DICOM import job started with ID: { lv_job_id }.| TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
```

```
MESSAGE 'Service quota exceeded.' TYPE 'I'.  
CATCH /aws1/cx_migthrottlingex.  
MESSAGE 'Request throttled.' TYPE 'I'.  
CATCH /aws1/cx_migvalidationex.  
MESSAGE 'Validation error.' TYPE 'I'.  
ENDTRY.
```

- Para obtener más información sobre la API, consulte [Start DICOMImport Job](#) in AWS SDK para obtener una referencia sobre la API ABAP de SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **TagResource** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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/1234567890123456789012/
imageset/18f88ac7870584f58d56256646b4d92b" \
  --tags '{"Deployment":"Development"}'
```

Este comando no genera ninguna salida.

Para obtener más información, consulta Cómo [etiquetar los recursos AWS HealthImaging](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [TagResource](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

        medicalImagingClient.tagResource(tagResourceRequest);

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

```

        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

```

- Para obtener más información sobre la API, consulte [TagResource](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

```

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

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

```

```
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    }
//  }

return response;
};
```

- Para obtener más información sobre la API, consulte [TagResource](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

```

        err.response["Error"]["Message"],
    )
    raise

```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

```

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

```

- Para obtener más información sobre la API, consulta [TagResource](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```

TRY.
    " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
    lo_mig->tagresource(
        iv_resourcearn = iv_resource_arn
        it_tags = it_tags ).
    MESSAGE 'Resource tagged successfully.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresource_notfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.

```

```
ENDTRY.
```

- Para obtener más información sobre la API, consulte [TagResource](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **UntagResource** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran 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/1234567890123456789012/  
imageset/18f88ac7870584f58d56256646b4d92b" \  
  --tag-keys '["Deployment"]'
```

Este comando no genera ninguna salida.

Para obtener más información, consulta Cómo [etiquetar los recursos AWS HealthImaging](#) en la Guía para AWS HealthImaging desarrolladores.

- Para obtener más información sobre la API, consulte [UntagResource](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

```
public static void untagMedicalImagingResource(MedicalImagingClient  
medicalImagingClient,  
    String resourceArn,  
    Collection<String> tagKeys) {  
    try {  
        UntagResourceRequest untagResourceRequest =  
        UntagResourceRequest.builder()  
            .resourceArn(resourceArn)  
            .tagKeys(tagKeys)  
            .build();  
  
        medicalImagingClient.untagResource(untagResourceRequest);  
  
        System.out.println("Tags have been removed from the resource.");  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

```
    }
  }
```

- Para obtener más información sobre la API, consulte [UntagResource](#) la Referencia AWS SDK for Java 2.x de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

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

```
//    }
// }

return response;
};
```

- Para obtener más información sobre la API, consulte [UntagResource](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

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

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

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

```
raise
```

El siguiente código crea una instancia del MedicalImagingWrapper objeto.

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

- Para obtener más información sobre la API, consulta [UntagResource](#) la AWS Referencia de API de SDK for Python (Boto3).

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
    lo_mig->untagresource(
        iv_resourcearn = iv_resource_arn
        it_tagkeys = it_tag_keys ).
    MESSAGE 'Resource untagged successfully.' TYPE 'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Resource not found.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [UntagResource](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **UpdateImageSetMetadata** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar UpdateImageSetMetadata.

CLI

AWS CLI

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

El siguiente ejemplo de `update-image-set-metadata` 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: eliminar un atributo de los metadatos del conjunto de imágenes

El siguiente ejemplo de `update-image-set-metadata` 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: eliminar una instancia de los metadatos del conjunto de imágenes

El siguiente ejemplo de `update-image-set-metadata` 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 \
  --force

```

Contenido de `metadata-updates.json`

```

{
  "DICOMUpdates": {
    "removableAttributes": "{\\"SchemaVersion\\": 1.1,\\"Study\\": {\\"Series\\": {\\"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\\": {\\"Instances\\": {\\"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\\": {}}}}}}"
  }
}

```

Salida:

```

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

```

Ejemplo 4: 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: añadir un elemento de datos DICOM privado a una instancia

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

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

```

": {"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {"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 un SOPInstance UID con el parámetro force

El siguiente `update-image-set-metadata` ejemplo muestra cómo actualizar un SOPInstance UID mediante el parámetro `force` para anular las restricciones de los metadatos del DICOM.

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

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 la AWS HealthImaging Guía para desarrolladores.

- Para obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

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

```

String versionId,
MetadataUpdates

metadataUpdates,

boolean force) {

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

        UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

```

Caso de uso 1: Insertar o actualizar un atributo.

```

final String insertAttributes = ""
{
    "SchemaVersion": 1.1,
    "Study": {
        "DICOM": {
            "StudyDescription": "CT CHEST"
        }
    }
}
"";

MetadataUpdates metadataInsertUpdates = MetadataUpdates.builder()
    .dicomUpdates(DICOMUpdates.builder()
        .updateableAttributes(SdkBytes.fromByteBuffer(
            ByteBuffer.wrap(insertAttributes

```

```

.getBytes(StandardCharsets.UTF_8)))
                .build())
            .build();

        updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                versionid, metadataInsertUpdates, force);

```

Caso de uso n.º 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 n.º 3: eliminar una instancia.

```

        final String removeInstance = ""
        {
            "SchemaVersion": 1.1,
            "Study": {
                "Series": {

```

```
"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {  
    "Instances": {  
  
        "1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}  
    }  
}  
}  
}  
}  
}  
}";  
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: Revertir 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 obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la Referencia AWS SDK for Java 2.x de la API.

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 * version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 200,
```

```

        //      requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
        //      extendedRequestId: undefined,
        //      cfId: undefined,
        //      attempts: 1,
        //      totalRetryDelay: 0
        // },
        //      createdAt: 2023-09-22T14:49:26.427Z,
        //      datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
        //      imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
        //      imageSetState: 'LOCKED',
        //      imageSetWorkflowStatus: 'UPDATING',
        //      latestVersionId: '4',
        //      updatedAt: 2023-09-27T19:41:43.494Z
        // }
    return response;
} catch (err) {
    console.error(err);
}
};

```

Caso de uso 1: Insertar o actualizar un atributo y forzar 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: Revertir a una versión anterior.

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

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

- Para obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la Referencia de AWS SDK para JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

```

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

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

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

```

El siguiente código crea una instancia del `MedicalImagingWrapper` objeto.

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

Caso de uso 1: Insertar o actualizar 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 n.º 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": {

"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                    "Instances": {

"1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                        }
                    }
                }
            }
        }"""
        metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

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

```

Caso de uso 4: Revertir a una versión anterior.

```

        metadata = {"revertToVersionId": "1"}

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

```

- Para obtener más información sobre la API, consulta [UpdateImageSetMetadata](#) la AWS Referencia de API de SDK for Python (Boto3).

 Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SAP ABAP

SDK para SAP ABAP

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    " iv_latest_version_id = '1'
    " iv_force = abap_false
    oo_result = lo_mig->updateimagesetmetadata(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id
        iv_latestversionid = iv_latest_version_id
        io_updateimagesetmetupdates = io_metadata_updates
        iv_force = iv_force ).
    DATA(lv_new_version) = oo_result->get_latestversionid( ).
    MESSAGE |Image set metadata updated to version: { lv_new_version }.| TYPE
'I'.
CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
CATCH /aws1/cx_migservicequotaexcdex.
    MESSAGE 'Service quota exceeded.' TYPE 'I'.
CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- Para obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la referencia sobre la API ABAP del AWS SDK para SAP.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus 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 varias funciones internas HealthImaging o combinadas con otros Servicios de AWS. En cada escenario se incluye un enlace al código fuente completo, con instrucciones de configuración y ejecución del código.

Los escenarios requieren un nivel intermedio de experiencia para ayudarlo a entender las acciones de servicio en su contexto.

Ejemplos

- [Comience con los conjuntos HealthImaging de imágenes y los marcos de imágenes mediante un AWS SDK](#)
- [Etiquetar un almacén HealthImaging de datos mediante un SDK AWS](#)
- [Etiquetar un conjunto HealthImaging de imágenes mediante un SDK AWS](#)

Comience con los conjuntos HealthImaging de imágenes y los marcos de imágenes mediante un AWS SDK

Los siguientes ejemplos de código muestran cómo importar archivos DICOM y descargar marcos de imágenes en HealthImaging ellos.

La implementación está estructurada como una aplicación de línea de comandos.

- Configure los recursos para una tarea de importación DICOM.
- Importe los archivos DICOM en un almacén de datos.
- Recupera el conjunto de imágenes IDs para el trabajo de importación.
- Recupera el marco de imágenes IDs de los conjuntos de imágenes.
- Descargue, decodifique y verifique los marcos de imágenes.
- Eliminación de recursos.

C++

SDK para C++

Crea una 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 los archivos DICOM en el bucket 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 archivos DICOM en el 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 que ha creado el trabajo de importación DICOM.

```
bool
AwsDoc::Medical_Imaging::getImageSetsForDicomImportJob(const Aws::String
&datastoreId,
                                                    const Aws::String
&importJobId,
                                                    Aws::Vector<Aws::String>
&imageSets,
                                                    const
Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome getDicomImportJobOutcome
= getDICOMImportJob(
    datastoreID, importJobId, clientConfiguration);
    bool result = false;
    if (getDicomImportJobOutcome.IsSuccess()) {
        auto outputURI =
getDicomImportJobOutcome.GetResult().GetJobProperties().GetOutputS3Uri();
        Aws::Http::URI uri(outputURI);
        const Aws::String &bucket = uri.GetAuthority();
        Aws::String key = uri.GetPath();

        Aws::S3::S3Client s3Client(clientConfiguration);
        Aws::S3::Model::GetObjectRequest objectRequest;
        objectRequest.SetBucket(bucket);
        objectRequest.SetKey(key + "/" + IMPORT_JOB_MANIFEST_FILE_NAME);

        auto getObjectOutcome = s3Client.GetObject(objectRequest);
        if (getObjectOutcome.IsSuccess()) {
            auto &data = getObjectOutcome.GetResult().GetBody();

            std::stringstream stringStream;
            stringStream << data.rdbuf();

            try {
                // Use JMESPath to extract the image set IDs.
                // https://jmespath.org/specification.html
                std::string jmesPathExpression =
"jobSummary.imageSetsSummary[].imageSetId";
                jsoncons::json doc = jsoncons::json::parse(stringStream.str());
```

```

        jsoncons::json imageSetsJson = jsoncons::jmespath::search(doc,
jmesPathExpression);\
        for (auto &imageSet: imageSetsJson.array_range()) {
            imageSets.push_back(imageSet.as_string());
        }

        result = true;
    }
    catch (const std::exception &e) {
        std::cerr << e.what() << '\n';
    }

}
else {
    std::cerr << "Failed to get object because "
        << getObjectOutcome.GetError().GetMessage() << std::endl;
}

}
else {
    std::cerr << "Failed to get import job status because "
        << getDicomImportJobOutcome.GetError().GetMessage() <<
std::endl;
}

    return result;
}

```

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,
                                                    const
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();
}

```

Descargar, decodificar y verificar los fotogramas 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 obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para C++ .
 - [DeleteImageSet](#)
 - [Consigue un DICOMImport trabajo](#)
 - [GetImageFrame](#)
 - [GetImageSetMetadata](#)
 - [SearchImageSets](#)
 - [Iniciar DICOMImport trabajo](#)

 Note

Hay más en marcha GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para JavaScript (v3)

Organizar los pasos (index.js).

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
```

```
    outputImageFrameIds,
  } from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
```

```

        decodeAndVerifyImages,
        saveState,
    ],
    context,
),
destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
    parseScenarioArgs(scenarios, {
        name: "Health Imaging Workflow",
        description:
            "Work with DICOM images using an AWS Health Imaging data store.",
        synopsis:
            "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    });
}

```

Implementar los recursos (deploy-steps.js).

```

import fs from "node:fs/promises";
import path from "node:path";

import {
    CloudFormationClient,
    CreateStackCommand,
    DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
    ScenarioAction,
    ScenarioInput,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

```

```
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
```

```

    async (** @type {} */ state) => {
      const stackName = state.getStackName;
      const datastoreName = state.getDatastoreName;
      const accountId = state.accountId;

      const command = new CreateStackCommand({
        StackName: stackName,
        TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
        Capabilities: ["CAPABILITY_IAM"],
        Parameters: [
          {
            ParameterKey: "datastoreName",
            ParameterValue: datastoreName,
          },
          {
            ParameterKey: "userAccountID",
            ParameterValue: accountId,
          },
        ],
      });

      const response = await cfnClient.send(command);
      state.stackId = response.StackId;
    },
    { skipWhen: (** @type {} */ state) => !state.deployStack },
  );

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (** @type {} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {

```

```

        acc[output.OutputKey] = output.OutputValue;
        return acc;
      }, {}));
    } else {
      throw new Error(
        `Stack creation failed with status: ${stack.StackStatus}`,
      );
    }
  });
},
{
  skipWhen: (/** @type {} */ state) => !state.deployStack,
},
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
     string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
`;
  },
  { skipWhen: (/** @type {} */ state) => !state.deployStack },
);

```

Copiar los archivos DICOM (dataset-steps.js).

```

import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,

```

```
ScenarioInput,
ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  }
);
```

```
    },
  );

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = "input/";
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey}
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });
  });
```

```

    const results = await Promise.all(copyPromises);
    state.copied0bjects = results.length;
  },
);

export const outputCopied0bjects = new ScenarioOutput(
  "outputCopied0bjects",
  (state) => `${state.copied0bjects} DICOM files were copied.`,
);

```

Iniciar la importación en el almacén de datos (import-steps.js).

```

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

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

```

```

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreId,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreId,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

```

```
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

Obtenga el conjunto de imágenes IDs (image-set-steps.js-).

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
 * [] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
```

```

        Key: key,
      });

      const response = await s3Client.send(command);
      const manifestContent = await response.Body.transformToString();
      state.manifestContent = JSON.parse(manifestContent);
    },
  );

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  /** @type {State} */ state => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce((ids, next) => {
        return Object.assign({}, ids, {
          [next.imageSetId]: next.imageSetId,
        });
      }, {});
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  /** @type {State} */ state =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}` ,
);

```

Obtenga el marco de la imagen IDs (image-frame-steps.js).

```

import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node:util";

import {
  ScenarioAction,
  ScenarioOutput,

```

```
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */
```

```
/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);
```

```

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  /** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
  "\n",
)}\n\n`;
    }

    return output;
  },
);

```

Comprobar los fotogramas de imágenes (verify-steps.js). Para la [verificación se utilizó la biblioteca de verificación de datos de AWS HealthImaging 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 [seriesInstanceId, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceId, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceId}
and sop ${sopInstanceId}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
```

```

        seriesInstanceId,
        sopInstanceId,
    ],
    { stdio: "inherit" },
);

await new Promise((resolve, reject) => {
    child.on("exit", (code) => {
        if (code === 0) {
            resolve();
        } else {
            reject(
                new Error(
                    `Verification tool exited with code ${code} for image set
${imageSetId}`,
                ),
            );
        }
    });
});
}
}
},
);
);

```

Destruir los recursos (clean-up-steps.js).

```

import {
    CloudFormationClient,
    DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
    MedicalImagingClient,
    DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
    ScenarioAction,
    ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

```

```
/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
```

```

    * ImageSetID: string,
    * Patient: Patient,
    * Study: Study
    * }} ImageSetMetadata
    */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  }
);

```

```
    }  
  },  
  {  
    skipWhen: (/** @type {} */ state) => !state.confirmCleanup,  
  },  
);  
  
export const deleteStack = new ScenarioAction(  
  "deleteStack",  
  async (/** @type {State} */ state) => {  
    const stackName = state.getStackName;  
  
    const command = new DeleteStackCommand({  
      StackName: stackName,  
    });  
  
    await cfnClient.send(command);  
    console.log(`Stack ${stackName} deletion initiated`);  
  },  
  {  
    skipWhen: (/** @type {} */ state) => !state.confirmCleanup,  
  },  
);
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript .
 - [DeleteImageSet](#)
 - [Consigue un DICOMImport trabajo](#)
 - [GetImageFrame](#)
 - [GetImageSetMetadata](#)
 - [SearchImageSets](#)
 - [Iniciar DICOMImport trabajo](#)

 Note

Hay más en marcha GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para Python (Boto3)

Crea una 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(
        "../../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml"
    ) as setup_file:
        setup_template = setup_file.read()
    print(f"\t\tCreating {stack_name}.")
    stack = self.cf_resource.create_stack(
        StackName=stack_name,
        TemplateBody=setup_template,
        Capabilities=["CAPABILITY_NAMED_IAM"],
        Parameters=[
            {
                "ParameterKey": "datastoreName",
                "ParameterValue": data_store_name,
            },
            {
                "ParameterKey": "userAccountID",
                "ParameterValue": account_id,
            },
        ],
    )
```

```

        print("\t\tWaiting for stack to deploy. This typically takes a minute or
two.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
        waiter.wait(StackName=stack.name)
        stack.load()
        print(f"\t\tStack status: {stack.stack_status}")

        outputs_dictionary = {
            output["OutputKey"]: output["OutputValue"] for output in
stack.outputs
        }
        self.input_bucket_name = outputs_dictionary["BucketName"]
        self.output_bucket_name = outputs_dictionary["BucketName"]
        self.role_arn = outputs_dictionary["RoleArn"]
        self.data_store_id = outputs_dictionary["DatastoreID"]
        return stack

```

Copie los archivos DICOM en el bucket 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 archivos DICOM en el almacén de datos de Amazon S3.

```

class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")

```

```

s3_client = boto3.client("s3")
return cls(medical_imaging_client, s3_client)

def start_dicom_import_job(
    self,
    data_store_id,
    input_bucket_name,
    input_directory,
    output_bucket_name,
    output_directory,
    role_arn,
):
    """
    Routine which starts a HealthImaging import job.

    :param data_store_id: The HealthImaging data store ID.
    :param input_bucket_name: The name of the Amazon S3 bucket containing the
DICOM files.
    :param input_directory: The directory in the S3 bucket containing the
DICOM files.
    :param output_bucket_name: The name of the S3 bucket for the output.
    :param output_directory: The directory in the S3 bucket to store the
output.
    :param role_arn: The ARN of the IAM role with permissions for the import.
    :return: The job ID of the import.
    """

    input_uri = f"s3://{input_bucket_name}/{input_directory}/"
    output_uri = f"s3://{output_bucket_name}/{output_directory}/"
    try:
        job = self.medical_imaging_client.start_dicom_import_job(
            jobName="examplejob",
            datastoreId=data_store_id,
            dataAccessRoleArn=role_arn,
            inputS3Uri=input_uri,
            outputS3Uri=output_uri,
        )
    except ClientError as err:
        logger.error(
            "Couldn't start DICOM import job. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )

```

```

        raise
    else:
        return job["jobId"]

```

Obtenga los conjuntos de imágenes que ha creado el trabajo de importación DICOM.

```

class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
        """
        Retrieves the image sets created for an import job.

        :param datastore_id: The HealthImaging data store ID
        :param import_job_id: The import job ID
        :return: List of image set IDs
        """

        import_job = self.medical_imaging_client.get_dicom_import_job(
            datastoreId=datastore_id, jobId=import_job_id
        )

        output_uri = import_job["jobProperties"]["outputS3Uri"]

```

```

        bucket = output_uri.split("/")[2]
        key = "/" + ".join(output_uri.split("/")[3:])

    # Try to get the manifest.
    retries = 3
    while retries > 0:
        try:
            obj = self.s3_client.get_object(
                Bucket=bucket, Key=key + "job-output-manifest.json"
            )
            body = obj["Body"]
            break
        except ClientError as error:
            retries = retries - 1
            time.sleep(3)
    try:
        data = json.load(body)
        expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
        image_sets = expression.search(data)
    except json.decoder.JSONDecodeError as error:
        image_sets = import_job["jobProperties"]

    return image_sets

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

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

```

```

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

```

Obtenga información sobre los marcos de imágenes para los conjuntos de imágenes.

```

class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

    def get_image_frames_for_image_set(self, datastore_id, image_set_id,
out_directory):
        """
        Get the image frames for an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.

```

```

:param out_directory: The directory to save the file.
:return: The image frames.
"""
image_frames = []
file_name = os.path.join(out_directory,
f"{image_set_id}_metadata.json.gzip")
file_name = file_name.replace("/", "\\")
self.get_image_set_metadata(file_name, datastore_id, image_set_id)
try:
    with gzip.open(file_name, "rb") as f_in:
        doc = json.load(f_in)
        instances = jmespath.search("Study.Series.*.Instances[].*[]", doc)
        for instance in instances:
            rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
            rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)

            image_frames_json = jmespath.search("ImageFrames[]{}", instance)
            for image_frame in image_frames_json:
                checksum_json = jmespath.search(
                    "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",

                    image_frame,
                )
                image_frame_info = {
                    "imageSetId": image_set_id,
                    "imageFrameId": image_frame["ID"],
                    "rescaleIntercept": rescale_intercept,
                    "rescaleSlope": rescale_slope,
                    "minPixelValue": image_frame["MinPixelValue"],
                    "maxPixelValue": image_frame["MaxPixelValue"],
                    "fullResolutionChecksum": checksum_json["Checksum"],
                }
                image_frames.append(image_frame_info)
    return image_frames
except TypeError:
    return {}
except ClientError as err:
    logger.error(
        "Couldn't get image frames for image set. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
return image_frames

```

```
def get_image_set_metadata(
    self, metadata_file, datastore_id, image_set_id, version_id=None
):
    """
    Get the metadata of an image set.

    :param metadata_file: The file to store the JSON gzipped metadata.
    :param datastore_id: The ID of the data store.
    :param image_set_id: The ID of the image set.
    :param version_id: The version of the image set.
    """

    try:
        if version_id:
            image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                versionId=version_id,
            )
        else:
            image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
        with open(metadata_file, "wb") as f:
            for chunk in
image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                if chunk:
                    f.write(chunk)

    except ClientError as err:
        logger.error(
            "Couldn't get image metadata. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

Descargar, decodificar y verificar los fotogramas de imágenes.

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

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

        :param file_path_to_write: The path to write the image frame's HTJ2K
        encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.medical_imaging_client.get_image_frame(
                datastoreId=datastore_id,
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    f.write(chunk)
        except ClientError as err:
```

```

        logger.error(
            "Couldn't get image frame. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

def download_decode_and_check_image_frames(
    self, data_store_id, image_frames, out_directory
):
    """
    Downloads image frames, decodes them, and uses the checksum to validate
    the decoded images.

    :param data_store_id: The HealthImaging data store ID.
    :param image_frames: A list of dicts containing image frame information.
    :param out_directory: A directory for the downloaded images.
    :return: True if the function succeeded; otherwise, False.
    """
    total_result = True
    for image_frame in image_frames:
        image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
        self.get_pixel_data(
            image_file_path,
            data_store_id,
            image_frame["imageSetId"],
            image_frame["imageFrameId"],
        )

        image_array = self.jph_image_to_opj_bitmap(image_file_path)
        crc32_checksum = image_frame["fullResolutionChecksum"]
        # Verify checksum.
        crc32_calculated = zlib.crc32(image_array)
        image_result = crc32_checksum == crc32_calculated
        print(
            f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
{image_result} "
        )
        total_result = total_result and image_result
    return total_result

    @staticmethod

```

```
def jph_image_to_opj_bitmap(jph_file):
    """
    Decode the image to a bitmap using an OPENJPEG library.
    :param jph_file: The file to decode.
    :return: The decoded bitmap as an array.
    """
    # Use format 2 for the JPH file.
    params = openjpeg.utils.get_parameters(jph_file, 2)
    print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

    image_array = openjpeg.utils.decode(jph_file, 2)

    return image_array
```

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 opout in stack.outputs:
        if opout["OutputKey"] == "DatastoreID":
            data_store_id = opout["OutputValue"]
    if data_store_id is not None:
        print(f"\t\tDeleting image sets in data store {data_store_id}.")
        image_sets = self.medical_imaging_wrapper.search_image_sets(
            data_store_id, {}
        )
        image_set_ids = [image_set["imageSetId"] for image_set in image_sets]

        for image_set_id in image_set_ids:
            self.medical_imaging_wrapper.delete_image_set(
                data_store_id, image_set_id
```

```

        )
        print(f"\t\tDeleted image set with id : {image_set_id}")

    print(f"\t\tDeleting {stack.name}.")
    stack.delete()
    print("\t\tWaiting for stack removal. This may take a few minutes.")
    waiter = self.cf_resource.meta.client.get_waiter("stack_delete_complete")
    waiter.wait(StackName=stack.name)
    print("\t\tStack delete complete.")

class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)

    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.medical_imaging_client.get_paginator("search_image_sets")

```

```

        page_iterator = paginator.paginate(
            datastoreId=datastore_id, searchCriteria=search_filter
        )
        metadata_summaries = []
        for page in page_iterator:
            metadata_summaries.extend(page["imageSetsMetadataSummaries"])
    except ClientError as err:
        logger.error(
            "Couldn't search image sets. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return metadata_summaries

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

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

```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
 - [DeleteImageSet](#)
 - [Consigue un DICOMImport trabajo](#)

- [GetImageFrame](#)
- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [Iniciar DICOMImport trabajo](#)

 Note

Hay más en marcha GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Etiquetar un almacén HealthImaging de datos mediante un SDK AWS

Los siguientes ejemplos de código muestran cómo etiquetar un almacén HealthImaging de datos.

Java

SDK para 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 obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para 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 obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript .
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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 MedicalImagingWrapper objeto.

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

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Etiquetar un conjunto HealthImaging de imágenes mediante un SDK AWS

Los siguientes ejemplos de código muestran cómo etiquetar un conjunto HealthImaging de imágenes.

Java

SDK para 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 obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

JavaScript

SDK para 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 obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript .
 - [ListTagsForResource](#)
 - [TagResource](#)

- [UntagResource](#)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Python

SDK para 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 MedicalImagingWrapper objeto.

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

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Uso de este servicio con un AWS SDK](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

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 CloudTrail captura 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, puedes CloudWatch hacer un seguimiento del uso de la CPU u otras métricas de tus EC2 instancias de Amazon 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 Software-as-a-Service (SaaS) y AWS 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](#)
- [Uso de Amazon CloudWatch con HealthImaging](#)
- [Uso de Amazon EventBridge con HealthImaging](#)

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](#).

Creating a trail

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 Consola de administración de AWS, vaya al menú Lookup attributes, seleccione Event Source y `elijamedical-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](#)
- [Servicios e integraciones compatibles con CloudTrail](#)
- [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 aquellos APIs 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 nativas que figuran en la [Referencia de HealthImaging API de AWS](#) se clasifican como 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 .

DICOMweb Las acciones de la API WADO-RS se clasifican como eventos de datos CloudTrail, por lo que debes activarlas. Para obtener más información, consulte [Registro de eventos Recuperación de datos DICOM de HealthImaging de datos](#) en la Guía del usuario.AWS CloudTrail

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario le 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 [CloudTrailuserIdentityelemento](#).

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": "XXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
    "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
    "accountId": "123456789012",
    "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "XXXXXXXXXXXXXXXXXXXX",
        "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
        "accountId": "123456789012",
        "userName": "TestAccessRole"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-10-28T15:52:42Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2022-10-28T16:02:30Z",
  "eventSource": "medical-imaging.amazonaws.com",
  "eventName": "GetDICOMImportJob",
  "awsRegion": "us-east-1",
```

```

    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-
    Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync
    http/Apache cfg/retry-mode/standard",
    "requestParameters": {
        "jobId": "5d08d05d6aab2a27922d6260926077d4",
        "datastoreId": "12345678901234567890123456789012"
    },
    "responseElements": null,
    "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
    "eventID": "26307f73-07f4-4276-b379-d362aa303b22",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "824333766656",
    "eventCategory": "Management"
}

```

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

Las métricas se incluyen en todos los informes HealthImaging APIs.

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	Description (Descripción)
Conteo de llamadas	El número de llamadas a APIs. 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

Puedes obtener métricas HealthImaging con la Consola de administración de AWS AWS CLI, la o la CloudWatch API. Puede usar la CloudWatch API a través de uno de los kits de desarrollo de software de Amazon AWS (SDKs) o las herramientas de la 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 [CloudWatchconsola Consola de administración de AWS 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 [ImageSetWorkflowStatus](#) la 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" : "bbc4f3cccb4e4095a34170fddc19b13d",
    "datastoreName": "test",
    "datastoreStatus": "DELETED"
  }
}

```

Eventos del almacén de datos: descripciones de metadatos

Name	Tipo	Description (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.

Name	Tipo	Description (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

Name	Tipo	Description (Descripción)
version	cadena	La versión EventBridge del esquema de eventos.

Name	Tipo	Description (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.

Name	Tipo	Description (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 de 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/bbc4f3cccbac4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
```

```

    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "CREATED"
  }
}

```

Image Set Copying

Estado - **COPYING**

```

{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Copying",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "COPYING",
    "sourceImageSetArn": "arn:aws:medical-imaging:us-
west-2:147997158357:datastore/c381ee9b9ef34902a45b476dd7be068b/
imageset/0309de3674fd551fa7ddd2880b21f990"
  }
}

```

Image Set Copying With Read Only Access

Estado - **COPYING_WITH_READ_ONLY_ACCESS**

```

{
  "version": "0",

```

```

    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Image Set Copying With Read Only Access",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
    "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS"
    }
  }
}

```

Image Set Copied

Estado - **COPIED**

```

{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Copied",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "COPIED"
  }
}

```

Image Set Copy Failed

Estado - **COPY_FAILED**

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Copy Failed",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "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",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
```

```
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "UPDATING"
  }
}
```

Image Set Updated

Estado - **UPDATED**

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Updated",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "UPDATED"
  }
}
```

Image Set Update Failed

Estado - **UPDATE_FAILED**

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Update Failed",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
```

```

    "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
    "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATE_FAILED"
    }
  }
}

```

Image Set Deleting

Estado - **DELETING**

```

{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Deleting",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "DELETING"
  }
}

```

Image Set Deleted

Estado - **DELETED**

```

{

```

```
"version": "0",
"id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
"detail-type": "Image Set Deleted",
"source": "aws.medical-imaging",
"account": "111122223333",
"time": "2024-03-14T00:01:00Z",
"region": "us-west-2",
"resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
"detail": {
  "imagingVersion": "1.0",
  "isPrimary": true,
  "imageSetVersion": "1",
  "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
  "imagesetId": "5b3a711878c34d40e888253319388649",
  "imageSetState": "DELETED",
  "imageSetWorkflowStatus": "DELETED"
}
}
```

Eventos del conjunto de imágenes: descripciones de metadatos

Name	Tipo	Description (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.

Name	Tipo	Description (Descripción)
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.
detail.isPrimary	booleano	Indica si los datos importados se organizaron correctamente en la jerarquía gestionada o si hay conflictos de metadatos que deban resolverse.
detail.imageSetVersion	cadena	La versión del conjunto de imágenes se incrementará cuando se importe una instancia más de una vez. La última versión sobrescribirá cualquier versión anterior almacenada en un conjunto de imágenes principal.
detail.datastoreId	cadena	El ID del almacén de datos que generó el evento de cambio de estado.

Name	Tipo	Description (Descripción)
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. También es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y la normativa 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](#)
- [Identity and Access Management para AWS HealthImaging](#)
- [Validación de conformidad para AWS HealthImaging](#)
- [Seguridad de la infraestructura en AWS HealthImaging](#)
- [Creación de HealthImaging recursos de AWS con AWS CloudFormation](#)
- [AWS HealthImaging y puntos finales de VPC de interfaz \(\)AWS PrivateLink](#)
- [Importación multicuenta para AWS HealthImaging](#)
- [Resiliencia en AWS HealthImaging](#)

Protección de datos en AWS HealthImaging

El AWS [modelo](#) de se aplica a 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. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las [Preguntas frecuentes sobre la privacidad de datos](#). Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el [Modelo de responsabilidad compartida de AWS y GDPR](#) en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las 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:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Se utiliza SSL/TLS para comunicarse con AWS los recursos. Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre el uso de CloudTrail senderos para capturar AWS actividades, consulte [Cómo trabajar con CloudTrail senderos](#) en la Guía del AWS CloudTrail usuario.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utiliza servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger la información confidencial almacenada en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-3 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte [Estándar de procesamiento de la información federal \(FIPS\) 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 HealthImaging o Servicios de AWS utiliza la consola, la API o. AWS CLI AWS SDKs Cualquier dato que introduzca en etiquetas o campos

de formato libre utilizados para los nombres se pueden emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Temas

- [Cifrado de datos](#)
- [Privacidad del tráfico de red](#)

Cifrado de datos

Con AWS HealthImaging, puede añadir una capa de seguridad a sus datos en reposo 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 de los AWS servicios
- Opciones flexibles de administración de claves AWS Key Management Service, que incluyen la posibilidad de elegir entre AWS gestionar las claves de cifrado o mantener el control total sobre las suyas propias.
- AWS claves de AWS KMS cifrado propias
- Colas de mensajes cifrados para la transmisión de datos confidenciales mediante cifrado del servidor (SSE) para Amazon SQS

Además, le AWS 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

De forma predeterminada, HealthImaging cifra los datos de los clientes en reposo mediante una clave propiedad del servicio AWS Key Management Service . Si lo desea, puede configurar HealthImaging el cifrado de los datos en reposo mediante una AWS KMS clave simétrica gestionada por el cliente que usted cree, posea y gestione. Para obtener más información, consulte [Crear una clave KMS de cifrado simétrico](#) en la Guía para desarrolladores.AWS Key Management Service

Cifrado en tránsito

HealthImaging utiliza TLS 1.2 para cifrar los datos en tránsito a través del punto final público y a través de los servicios de backend.

Administración de claves

AWS KMS las claves (claves KMS) son el recurso principal de AWS Key Management Service. También puede generar claves de datos para usarlas fuera de AWS KMS.

AWS clave KMS propia

HealthImaging utiliza estas claves de forma predeterminada para cifrar automáticamente la información potencialmente confidencial, como los datos de identificación personal o de información de salud privada (PHI) en reposo. AWS las claves KMS propias no se almacenan en su cuenta. Forman parte de una colección de claves de KMS que AWS posee y administra para su uso en varias AWS cuentas. AWS los servicios pueden usar claves KMS AWS propias para proteger sus datos. No puede ver, administrar, usar las claves AWS de KMS propias ni auditar 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 cobrará una cuota mensual ni una cuota de uso si utilizas claves de KMS AWS propias, y estas no se descontarán de AWS KMS las cuotas de tu cuenta. Para más información, consulte las [claves propias de AWS](#) en la Guía para desarrolladores de AWS Key Management Service .

Clave de KMS administradas por el cliente

Si quieres tener el control total sobre el AWS KMS ciclo de vida y el uso, HealthImaging admite el uso de una clave KMS simétrica administrada por el cliente que tú crees, poseas y administres. Como usted tiene el control total de dicho cifrado, puede realizar tareas como las siguientes:

- Establecer y mantener políticas de claves, políticas de IAM y concesiones
- Rotar el material criptográfico
- Habilitar y deshabilitar políticas de claves
- Adición de etiquetas
- Crear alias de clave
- Programar la eliminación de claves

También puedes utilizarla CloudTrail para realizar un seguimiento de las solicitudes que se HealthImaging envían AWS KMS en tu nombre. Se aplican AWS KMS cargos adicionales. Para obtener más información, consulta [las claves administradas por el cliente](#) en la Guía para AWS Key Management Service desarrolladores.

Creación de claves administradas por el cliente

Puede crear una clave simétrica gestionada por el cliente mediante el Consola de administración de AWS o el AWS KMS APIs. Para más información, consulte [Creación de claves de KMS de cifrado simétricas](#) en la Guía para desarrolladores de AWS Key Management Service .

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 [Administración del acceso a las claves](#) en la Guía para desarrolladores de AWS Key Management Service .

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 clave de KMS específica, lo que permite al usuario acceder a las [operaciones de subvención](#) HealthImaging requeridas. Para más información, consulte [Concesiones en AWS KMS](#) en la Guía para desarrolladores de AWS Key Management Service .

Para utilizar la clave de KMS gestionada por el cliente con sus HealthImaging recursos, la política de claves debe permitir las siguientes operaciones de API:

- `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",
        "kms:GenerateDataKeyWithoutPlaintext"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:EncryptionContext:kms-arn": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f",
            "kms:EncryptionContext:aws:medical-imaging:datastoreId": "datastoreId"
        }
    }
}

```

Permisos de IAM necesarios cuando al utilizar una clave de KMS administrada por el cliente

Al crear un banco de datos con el AWS KMS cifrado habilitado mediante una clave de KMS administrada por el cliente, se requieren permisos tanto para la política de claves como para la política de IAM para el usuario o rol que crea el banco de HealthImaging datos.

Para más información acerca de las políticas de claves, consulte [Habilitación de las políticas de IAM](#) en la Guía para desarrolladores de AWS Key Management Service .

El usuario de IAM, el rol de IAM o la AWS cuenta que crea los repositorios debe tener permisos para la política siguiente, además de los permisos necesarios para AWS. HealthImaging

¿Cómo se utilizan HealthImaging las subvenciones en AWS KMS

HealthImaging requiere una [concesión](#) para usar la clave de KMS gestionada por el cliente. Al crear un almacén de datos cifrado con una clave KMS gestionada por el cliente, HealthImaging crea una concesión en tu nombre enviando una [CreateGrants](#) solicitud a AWS KMS. Las concesiones in se AWS KMS utilizan para dar HealthImaging acceso a una clave KMS de la cuenta de un cliente.

Las subvenciones que se HealthImaging crean en tu nombre no se deben revocar ni retirar. Si revoca o retira la concesión que HealthImaging autoriza el uso de las AWS KMS claves de su cuenta, HealthImaging no puede acceder a estos datos, cifra los nuevos recursos de imágenes introducidos en el almacén de datos o los descifra al extraerlos. 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.

Monitorización de las claves de cifrado para HealthImaging

Puede utilizarla CloudTrail para realizar un seguimiento de las solicitudes que se HealthImaging envían AWS KMS en su nombre cuando utiliza una clave KMS administrada 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` `GenerateDataKeyDecrypt`, y para supervisar AWS KMS las operaciones solicitadas `DescribeKey` para acceder HealthImaging 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 clave de KMS proporcionada por el cliente, lo que HealthImaging permite utilizar esa clave de KMS 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. Las subvenciones AWS KMS se utilizan para dar HealthImaging acceso a una AWS KMS clave de la cuenta de un cliente.

```
{
  "KeyId": "arn:aws:kms:us-east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
  "GrantId":
    "44e88bc45b769499ce5ec4abd5ecb27eeb3b178a4782452aae65fe885ee5ba20",
  "Name": "MedicalImagingGrantForQID0_ebffa634a-2d16-4046-9238-e3dc4ab54d29",
  "CreationDate": "2025-04-17T20:12:49+00:00",
  "GranteePrincipal": "AWS Internal",
  "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
  "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
  "Operations": [
    "Decrypt",
    "Encrypt",
    "GenerateDataKey",
    "GenerateDataKeyWithoutPlaintext",
    "ReEncryptFrom",
    "ReEncryptTo",
    "CreateGrant",
```

```

        "RetireGrant",
        "DescribeKey"
    ]
},
{
    "KeyId": "arn:aws:kms:us-east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
    "GrantId":
"9e5fd5ba7812daf75be4a86efb2b1920d6c0c9c0b19781549556bf2ff98953a1",
    "Name": "2025-04-17T20:12:38",
    "CreationDate": "2025-04-17T20:12:38+00:00",
    "GranteePrincipal": "medical-imaging.us-east-1.amazonaws.com",
    "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
    "IssuingAccount": "AWS Internal",
    "Operations": [
        "Decrypt",
        "Encrypt",
        "GenerateDataKey",
        "GenerateDataKeyWithoutPlaintext",
        "ReEncryptFrom",
        "ReEncryptTo",
        "CreateGrant",
        "RetireGrant",
        "DescribeKey"
    ]
},
{
    "KeyId": "arn:aws:kms:us-east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
    "GrantId":
"ab4a9b919f6ca8eb2bd08ee72475658ee76cfc639f721c9caaa3a148941bcd16",
    "Name": "9d060e5b5d4144a895e9b24901088ca5",
    "CreationDate": "2025-04-17T20:12:39+00:00",
    "GranteePrincipal": "AWS Internal",
    "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
    "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
    "Operations": [
        "Decrypt",
        "Encrypt",
        "GenerateDataKey",
        "GenerateDataKeyWithoutPlaintext",
        "ReEncryptFrom",
        "ReEncryptTo",
        "DescribeKey"
    ]
}

```

```

    ],
    "Constraints": {
      "EncryptionContextSubset": {
        "kms-arn": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c"
      }
    }
  }
}

```

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 HealthImaging realizar 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"
}

```

El siguiente ejemplo muestra cómo se HealthImaging utiliza la `DescribeKey` operación para comprobar si la AWS KMS clave propiedad del AWS KMS cliente está en un estado utilizable y 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 recursos siguientes proporcionan más información sobre el cifrado de datos en reposo y se encuentran en la Guía para desarrolladores de AWS Key Management Service .

- [Conceptos de AWS KMS](#)
- [Prácticas recomendadas 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. El tráfico entre HTTPS HealthImaging y lo AWS Key Management Service usa 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. Debe permitir únicamente las conexiones cifradas a través de HTTPS (TLS) según las políticas de IAM de los buckets de Amazon [aws:SecureTransport condition](#)S3. 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 de la AWS red y se cifra mediante TLS.

Identity and Access Management para AWS HealthImaging

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a los AWS recursos. Los administradores de IAM controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. HealthImaging La IAM es una Servicio de AWS opción que puede utilizar sin coste adicional.

Temas

- [Público](#)

- [Autenticación con identidades](#)
- [Administración del acceso con políticas](#)
- [Cómo HealthImaging funciona AWS con IAM](#)
- [Ejemplos de políticas basadas en identidad para AWS HealthImaging](#)
- [AWS políticas administradas para AWS HealthImaging](#)
- [La multiservicio confundió la prevención policial en HealthImaging](#)
- [Solución de problemas de HealthImaging identidad y acceso a AWS](#)

Público

La forma de usar AWS Identity and Access Management (IAM) varía según la función que desempeñes:

- Usuario del servicio: solicite permisos al administrador si no puede acceder a las características (consulte [Solución de problemas de HealthImaging identidad y acceso a AWS](#)).
- Administrador del servicio: determine el acceso de los usuarios y envíe las solicitudes de permiso (consulte [Cómo HealthImaging funciona AWS con IAM](#)).
- Administrador de IAM: escribe las políticas para administrar el acceso (consulte [Ejemplos de políticas basadas en identidad para AWS HealthImaging](#)).

Autenticación con identidades

La autenticación es la forma en que inicias sesión AWS con tus credenciales de identidad. Debe autenticarse como usuario de Usuario raíz de la cuenta de AWS IAM o asumir una función de IAM.

Puede iniciar sesión como una identidad federada con las credenciales de una fuente de identidad, como AWS IAM Identity Center (IAM Identity Center), la autenticación de inicio de sesión único o las credenciales. Google/Facebook Para obtener más información sobre el inicio de sesión, consulte [Cómo iniciar sesión en la Cuenta de AWS](#) en la Guía del usuario de AWS Sign-In .

Para el acceso programático, AWS proporciona un SDK y una CLI para firmar criptográficamente las solicitudes. Para obtener más información, consulte [AWS Signature Version 4 para solicitudes de API](#) en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear un Cuenta de AWS, se comienza con una identidad de inicio de sesión denominada usuario Cuenta de AWS raíz que tiene acceso completo a todos Servicios de AWS los recursos. Se recomienda encarecidamente que no utilice el usuario raíz para las tareas diarias. Para ver las tareas que requieren credenciales de usuario raíz, consulte [Tareas que requieren credenciales de usuario raíz](#) en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos 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 empresarial, del proveedor de identidades web o al Directory Service que se accede Servicios de AWS mediante credenciales de una fuente de identidad. Las identidades federadas asumen roles que proporcionan credenciales temporales.

Para una administración de acceso centralizada, se recomienda AWS IAM Identity Center. Para obtener más información, consulte [¿Qué es el Centro de identidades de IAM?](#) en la Guía del usuario de AWS IAM Identity Center .

Usuarios y grupos de IAM

Un [usuario de IAM](#) es una identidad con permisos específicos para una sola persona o aplicación. Recomendamos el uso de credenciales temporales en lugar de usuarios de IAM con credenciales de larga duración. Para obtener más información, consulte [Exigir a los usuarios humanos que utilicen la federación con un proveedor de identidad para acceder AWS mediante credenciales temporales](#) en la Guía del usuario de IAM.

Un [grupo de IAM](#) especifica un conjunto de usuarios de IAM y facilita la administración de los permisos para grupos grandes de usuarios. Para obtener más información, consulte [Casos de uso para usuarios de IAM](#) en la Guía del usuario de IAM.

Roles de IAM

Un [Rol de IAM](#) es una identidad con permisos específicos que proporciona credenciales temporales. Puede asumir un rol [cambiando de un rol de usuario a uno de IAM \(consola\)](#) o llamando a una AWS CLI operación de AWS API. Para obtener más información, consulte [Métodos para asumir un rol](#) en la Guía del usuario de IAM.

Las funciones de IAM son útiles para el acceso de usuarios federados, los permisos de usuario de IAM temporales, el acceso entre cuentas, el acceso entre servicios y las aplicaciones que se ejecutan en Amazon. EC2 Para obtener más información, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Administración del acceso con políticas

AWS Para controlar el acceso, puede crear políticas y adjuntarlas a identidades o recursos. AWS Una política define los permisos cuando están asociados a una identidad o un recurso. AWS evalúa estas políticas cuando un director hace una solicitud. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre los documentos de políticas de JSON, consulte [Información general de políticas de JSON](#) en la Guía del usuario de IAM.

Mediante las políticas, los administradores especifican quién tiene acceso a qué, definiendo qué entidad principal puede realizar acciones sobre qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM crea políticas de IAM y las agrega a roles, que los usuarios pueden asumir posteriormente. Las políticas de IAM definen permisos independientemente del método que se utilice para realizar la operación.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de política de permisos JSON que asocia a una identidad (usuario, grupo o rol). Estas políticas controlan qué acciones pueden realizar las identidades, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en la identidad, consulte [Definición de permisos de IAM personalizados con políticas administradas por el cliente](#) en la Guía del usuario de IAM.

Las políticas basadas en identidad pueden ser políticas insertadas (incrustadas directamente en una sola identidad) o políticas administradas (políticas independientes asociadas a varias identidades). Para obtener información sobre cómo elegir entre políticas administradas e insertadas, consulte [Selección entre políticas administradas y políticas insertadas](#) en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de políticas JSON que se asocian a un recurso. Los ejemplos incluyen las Políticas de confianza de roles de IAM 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. Debe [especificar una entidad principal](#) en una política basada en recursos.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Otros tipos de políticas

AWS admite tipos de políticas adicionales que pueden establecer los permisos máximos que conceden los tipos de políticas más comunes:

- Límites de permisos: establecen los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM. Para obtener más información, consulte [Límites de permisos para las entidades de IAM](#) en la Guía del usuario de IAM.
- Políticas de control de servicios (SCPs): especifican los permisos máximos para una organización o unidad organizativa en AWS Organizations. Para obtener más información, consulte [Políticas de control de servicios](#) en la Guía del usuario de AWS Organizations .
- Políticas de control de recursos (RCPs): establece los permisos máximos disponibles para los recursos de tus cuentas. Para obtener más información, consulte [Políticas de control de recursos \(RCPs\)](#) en la Guía del AWS Organizations usuario.
- Políticas de sesión: políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal para un rol o un usuario federado. Para obtener más información, consulte [Políticas de sesión](#) en la Guía del usuario de IAM.

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 se AWS determina si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la [lógica de evaluación de políticas](#) en la Guía del usuario de IAM.

Cómo HealthImaging funciona AWS con IAM

Antes de usar IAM para administrar el acceso HealthImaging, conozca las funciones de IAM disponibles para su uso. HealthImaging

Características de IAM que puede usar con AWS HealthImaging

Característica de IAM	HealthImaging soporte
Políticas basadas en identidades	Sí
Políticas basadas en recursos	No
Acciones de políticas	Sí
Recursos de políticas	Sí
Claves de condición de política (específicas del servicio)	Sí
ACLs	No
ABAC (etiquetas en políticas)	Parcial
Credenciales temporales	Sí
Permisos de entidades principales	Sí
Roles de servicio	Sí
Roles vinculados al servicio	No

Para obtener una visión general de cómo HealthImaging funcionan otros AWS servicios con la mayoría de las funciones de IAM, consulte [AWS los servicios que funcionan con IAM](#) en la Guía del usuario de IAM.

Políticas basadas en la identidad para HealthImaging

Compatibilidad con las políticas basadas en identidad: sí

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, 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 más información sobre cómo crear una política basada en la identidad, consulte [Definición de permisos de IAM personalizados con políticas administradas por el cliente](#) en la Guía del usuario de IAM.

Con las políticas basadas en identidades de IAM, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. Para obtener más información sobre los elementos que puede utilizar en una política de JSON, consulte [Referencia de los elementos de la política de JSON de IAM](#) en la Guía del usuario de IAM.

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 identidad para AWS HealthImaging](#)

Políticas basadas en recursos incluidas HealthImaging

Admite políticas basadas en recursos: no

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Los ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM 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 basada en recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Para habilitar el acceso entre cuentas, puede especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Para obtener más información, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Acciones políticas para HealthImaging

Compatibilidad con las acciones de políticas: sí

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de HealthImaging acciones, consulte [Acciones definidas por 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 identidad para AWS HealthImaging](#)

Recursos de políticas para HealthImaging

Compatibilidad con los recursos de políticas: sí

Los administradores pueden usar las políticas de AWS JSON 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` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Como práctica recomendada, especifique un recurso utilizando el [Nombre de recurso de Amazon \(ARN\)](#). En el caso de las acciones que no admiten permisos por recurso, 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 correspondientes ARNs, consulte [Tipos de recursos definidos por AWS HealthImaging](#) en la Referencia de autorización de servicios. Para saber con qué acciones y recursos puede utilizar un ARN, consulte [Acciones definidas por AWS](#). HealthImaging

Para ver ejemplos de políticas HealthImaging basadas en la identidad, consulte. [Ejemplos de políticas basadas en 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 las políticas de AWS JSON 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` especifica cuándo se ejecutan las instrucciones en función de criterios definidos. Puede crear expresiones condicionales que utilizan [operadores de condición](#), tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud. Para ver todas las claves de condición AWS globales, consulte las claves de [contexto de condición AWS globales](#) en la Guía del usuario de IAM.

Para ver una lista de claves de HealthImaging condición, consulte [Claves de condición de AWS HealthImaging](#) en la Referencia de autorización de servicios. 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 identidad para AWS HealthImaging](#)

ACLs in HealthImaging

Soporta ACLs: 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. ACLs son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

RBAC con HealthImaging

Compatible con RBAC

Sí

El modelo de autorización tradicional utilizado en IAM se denomina control de acceso basado en roles (RBAC). RBAC define permisos en función de la función laboral de una persona, conocida fuera de AWS como rol. Para obtener más información, consulte [Comparación de ABAC con el modelo RBAC tradicional](#) en la Guía del usuario de IAM.

ABAC con HealthImaging

Compatibilidad con ABAC (etiquetas en las políticas): parcial

 Warning

ABAC no se aplica mediante la acción de la API `SearchImageSets`. 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 utilizar ABAC, es necesario que el conjunto de imágenes tenga la misma etiqueta que el almacén de datos. Para obtener más información, consulta [Etiquetado de recursos con AWS HealthImaging](#).

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define permisos en función de atributos denominados etiquetas. Puede adjuntar etiquetas a las entidades y AWS los recursos de IAM y, a continuación, diseñar políticas de ABAC para permitir las operaciones cuando la etiqueta del principal coincida con la etiqueta del recurso.

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 sobre ABAC, consulte [Definición de permisos con la autorización de ABAC](#) en la Guía del usuario de IAM. Para ver un tutorial con los pasos para configurar ABAC, consulte [Uso del control de acceso basado en atributos \(ABAC\)](#) en la Guía del usuario de IAM.

Utilizar credenciales temporales con HealthImaging

Compatibilidad con credenciales temporales: sí

Las credenciales temporales proporcionan acceso a AWS los recursos a corto plazo y se crean automáticamente cuando se utiliza la federación o se cambia de rol. 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](#) y [Servicios de AWS que funcionan con IAM](#) en la Guía del usuario de IAM.

Permisos principales entre servicios para HealthImaging

Admite sesiones de acceso directo (FAS): sí

Cuando utilizas un usuario o un rol de IAM para realizar acciones en él AWS, se te considera principal. 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 de AWS HealthImaging](#) en la Referencia de autorización de servicios.

Roles de servicio para HealthImaging

Compatible con roles de servicio: sí

Un rol de servicio es un [rol de IAM](#) que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. 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 afectar a 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 el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

Para más información sobre cómo crear o administrar roles vinculados a servicios, consulta [Servicios de AWS que funcionan con 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 identidad para AWS HealthImaging

De forma predeterminada, los usuarios y roles no tienen permiso para crear, ver ni modificar recursos de HealthImaging. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan.

Para obtener información acerca de cómo crear una política basada en identidades de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte [Creación de políticas de IAM \(consola\)](#) en la Guía del usuario de IAM.

Para obtener más información sobre las acciones y los tipos de recursos definidos por Awesome, incluido el ARNs formato de cada uno de los tipos de recursos, consulte [las claves de condición, recursos y acciones de AWS Awesome](#) en la referencia de autorización de servicios.

Temas

- [Prácticas recomendadas sobre las políticas](#)
- [Mediante la consola de HealthImaging](#)
- [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 generar costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas AWS administradas y avance hacia los permisos con privilegios mínimos: para empezar a conceder permisos a sus usuarios y cargas de trabajo, utilice las políticas AWS administradas que otorgan permisos para muchos casos de uso comunes. Están disponibles en su Cuenta de AWS. Le recomendamos que reduzca aún más los permisos definiendo políticas administradas por el AWS cliente que sean específicas para sus casos de uso. Con el fin de obtener más información, consulte las [políticas administradas por AWS](#) o las [políticas administradas por AWS para funciones de tarea](#) en la Guía de usuario de IAM.
- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, 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. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulte [Políticas y permisos en IAM](#) en la Guía del usuario de IAM.

- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puedes usar condiciones para conceder el acceso a las acciones del servicio si se utilizan a través de una acción específica Servicio de AWS, por ejemplo CloudFormation. Para obtener más información, consulte [Elementos de la política de JSON de IAM: Condición](#) en la Guía del usuario de IAM.
- Utiliza el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte [Validación de políticas con el Analizador de acceso de IAM](#) en la Guía del usuario de IAM.
- Requerir autenticación multifactor (MFA): si tiene un escenario que requiere usuarios de IAM o un usuario raíz en Cuenta de AWS su cuenta, active la MFA para mayor seguridad. Para exigir la MFA cuando se invoquen las operaciones de la API, añada condiciones de MFA a sus políticas. Para más información, consulte [Acceso seguro a la API con MFA](#) en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte [Prácticas recomendadas de seguridad en IAM](#) en la Guía del usuario de IAM.

Mediante la consola de HealthImaging

Para acceder a la HealthImaging consola de AWS, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle enumerar y ver detalles sobre los HealthImaging recursos de su cuenta Cuenta de AWS. Si crea una política basada en identidades que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

No es necesario que concedas permisos mínimos de consola a los usuarios que solo realicen llamadas a la API AWS CLI o a la AWS API. En su lugar, permita el acceso únicamente a las acciones que coincidan con la operación de API que intentan realizar.

Para garantizar que los usuarios y los roles puedan seguir utilizando la HealthImaging consola, adjunte también la política *ReadOnly* AWS gestionada HealthImaging *ConsoleAccess* o la política gestionada a las entidades. Para obtener más información, consulte [Adición de permisos a un usuario](#) en la Guía del usuario de IAM:

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para completar esta acción en la consola o mediante programación mediante la API AWS CLI o AWS .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

AWS políticas administradas para AWS HealthImaging

Una política AWS administrada es una política independiente creada y administrada por AWS. AWS Las políticas administradas están diseñadas para proporcionar permisos para muchos casos de uso comunes, de modo que pueda empezar a asignar permisos a usuarios, grupos y funciones.

Ten en cuenta que es posible que las políticas AWS administradas no otorguen permisos con privilegios mínimos para tus casos de uso específicos, ya que están disponibles para que los usen todos los AWS clientes. Se recomienda definir [políticas administradas por el cliente](#) específicas para sus casos de uso a fin de reducir aún más los permisos.

No puedes cambiar los permisos definidos en AWS las políticas administradas. Si AWS actualiza los permisos definidos en una política AWS administrada, la actualización afecta a todas las identidades principales (usuarios, grupos y roles) a las que está asociada la política. AWS es más probable que actualice una política AWS administrada cuando Servicio de AWS se lance una nueva o cuando estén disponibles nuevas operaciones de API para los servicios existentes.

Para obtener más información, consulte [Políticas administradas por AWS](#) en la Guía del usuario de IAM.

Temas

- [AWS política gestionada: AWSHealth ImagingFullAccess](#)
- [AWS política gestionada: AWSHealth ImagingReadOnlyAccess](#)
- [HealthImaging actualizaciones de las políticas gestionadas AWS](#)

AWS política gestionada: AWSHealth ImagingFullAccess

Puede asociar la política AWSHealthImagingFullAccess a las identidades de IAM.

Esta política concede permisos administrativos a todas HealthImaging las acciones.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "medical-imaging:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "medical-imaging.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS política gestionada: AWSHealth ImagingReadOnlyAccess

Puede asociar la política AWSHealthImagingReadOnlyAccess a las identidades de IAM.

Esta política otorga permisos de solo lectura a acciones específicas de AWS HealthImaging .

JSON

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

```
"Statement": [{
  "Effect": "Allow",
  "Action": [
    "medical-imaging:GetDICOMImportJob",
    "medical-imaging:GetDatastore",
    "medical-imaging:GetImageFrame",
    "medical-imaging:GetImageSet",
    "medical-imaging:GetImageSetMetadata",
    "medical-imaging:ListDICOMImportJobs",
    "medical-imaging:ListDatastores",
    "medical-imaging:ListImageSetVersions",
    "medical-imaging:ListTagsForResource",
    "medical-imaging:SearchImageSets"
  ],
  "Resource": "*"
}]
}
```

HealthImaging actualizaciones de las políticas gestionadas AWS

Consulte los detalles sobre las actualizaciones de las políticas AWS administradas HealthImaging desde que este servicio 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 rastrear los cambios	HealthImaging comenzó a realizar un seguimiento de los cambios de sus políticas AWS gestionadas.	19 de julio de 2023

La multiservicio confundió la prevención policial en HealthImaging

El problema de la sustitución confusa es un problema de seguridad en el que una entidad que no tiene permiso para realizar una acción puede obligar a una entidad con más privilegios a realizar la

acción. En AWS, la suplantación de identidad entre servicios puede provocar el confuso problema de un diputado. La suplantación entre servicios puede producirse cuando un servicio (el servicio que lleva a cabo las llamadas) llama a otro servicio (el servicio al que se llama). El servicio que lleva a cabo las llamadas se puede manipular para utilizar sus permisos a fin de actuar en función de los recursos de otro cliente de una manera en la que no debe tener permiso para acceder. Para evitarlo, AWS proporciona herramientas que le ayudan a proteger los datos de todos los servicios con directores de servicio a los que se les ha dado acceso a los recursos de su cuenta.

Le recomendamos que utilice las claves de contexto de condición [aws:SourceAccount](#) global [aws:SourceArn](#) las claves de contexto en sus políticas de relaciones de confianza de roles de `ImportJobDataAccessRole` IAM para limitar los permisos que AWS HealthImaging concede a otro servicio a su recurso. Utilice `aws:SourceArn` para asociar solo un recurso al acceso entre servicios. Utilice `aws:SourceAccount` para permitir que cualquier recurso de esa cuenta se asocie al uso entre servicios. Si se utilizan ambas claves de contexto de condición global, tanto el valor `aws:SourceAccount` como la cuenta a la que se hace referencia en el valor `aws:SourceArn` deben utilizar el mismo ID de cuenta cuando se utilicen en la misma instrucción de política.

El valor de `aws:SourceArn` debe ser el ARN del almacén de datos afectado. Si no conoce el ARN completo del banco de datos o si está especificando varios bancos de datos, utilice la clave de condición de contexto `aws:SourceArn` global con el comodín `*` para las partes desconocidas del ARN. Por ejemplo, puede establecer `aws:SourceArn` para `arn:aws:medical-imaging:us-west-2:111122223333:datastore/*`.

En el siguiente ejemplo de política de confianza, utilizamos la clave de `aws:SourceAccount` condición `aws:SourceArn` y para restringir el acceso a la entidad principal del servicio en función del ARN del almacén de datos a fin de evitar el confuso problema del diputado.

Solución de problemas de HealthImaging identidad y acceso a AWS

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas más comunes que pueden surgir al trabajar con un HealthImaging 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 mí accedan Cuenta de AWS 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.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM mateojackson intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *my-example-widget*, pero no tiene los permisos ficticios AWS: *GetWidget*.

```
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 necesita ayuda, póngase en contacto con su 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 recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a HealthImaging.

Algunos 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 la función al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado marymajor intenta utilizar la consola para realizar una acción en 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 mí accedan Cuenta de AWS a mis HealthImaging recursos

Se puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Se puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que respaldan las políticas basadas en recursos o las listas de control de acceso (ACLs), puedes usar esas políticas para permitir que las personas accedan a tus recursos.

Para obtener más información, consulte lo siguiente:

- Para saber si HealthImaging es compatible con estas funciones, consulte [Cómo HealthImaging funciona AWS con IAM](#)
- Para obtener información sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte [Proporcionar acceso a un usuario de IAM en otro usuario de su propiedad Cuenta de AWS en](#) la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta [Cómo proporcionar acceso a recursos que Cuentas de AWS son propiedad de terceros](#) en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulte [Proporcionar acceso a usuarios autenticados externamente \(identidad federada\)](#) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre las políticas basadas en roles y en recursos para el acceso entre cuentas, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Validación de conformidad para AWS HealthImaging

Los auditores externos evalúan la seguridad y la conformidad de AWS HealthImaging como parte de varios programas de AWS conformidad. Pues HealthImaging, esto incluye la HIPAA.

Para ver una lista de AWS los servicios incluidos en el ámbito de los programas de conformidad específicos, consulte [Servicios de AWS incluidos](#) . Para obtener información general, consulte Programas de [AWS cumplimiento > Programas AWS](#) .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte [Descarga de informes en AWS Artifact](#).

Su responsabilidad de conformidad al utilizar AWS HealthImaging viene determinada por la confidencialidad de sus datos, los objetivos de conformidad de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudarlo a cumplir con los requisitos:

- [AWS Soluciones para socios](#): las guías de implementación automatizadas de referencia para la seguridad y el cumplimiento de normas abordan las consideraciones arquitectónicas y proporcionan los pasos necesarios para implementar entornos básicos centrados en la seguridad y el cumplimiento. AWS
- Documento técnico sobre [cómo diseñar una arquitectura basada en la seguridad y el cumplimiento de la HIPAA](#): este documento técnico describe cómo pueden utilizar las empresas para crear aplicaciones que cumplan con la HIPAA. AWS
- [GxP Systems on AWS](#): este documento técnico proporciona información sobre cómo AWS abordar el cumplimiento y la seguridad relacionados con la GXP y proporciona orientación sobre el uso de los servicios en el contexto de la GxP. AWS
- [AWS Recursos de cumplimiento](#): esta colección de libros de trabajo y guías puede aplicarse a su sector y ubicación.
- [Evaluación de los recursos con reglas](#): AWS Config evalúa en qué medida las configuraciones de sus recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- [AWS Security Hub CSPM](#)— Este AWS servicio proporciona una visión integral del estado de su seguridad AWS que le ayuda a comprobar el cumplimiento de los estándares y las mejores prácticas del sector de la seguridad.

Seguridad de la infraestructura en AWS HealthImaging

Como servicio gestionado, AWS HealthImaging está protegido por los procedimientos de seguridad de la red AWS global que se describen en el documento técnico [Amazon Web Services: Overview of Security Processes](#).

Utiliza las llamadas a la API AWS publicadas para acceder a HealthImaging 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 HealthImaging recursos de AWS con AWS CloudFormation

AWS HealthImaging está integrado con AWS CloudFormation un servicio que le ayuda a modelar y configurar sus AWS recursos para que pueda dedicar menos tiempo a crear y administrar sus recursos e infraestructura. Cree una plantilla que describa todos los AWS recursos que desee y los CloudFormation aprovisiona y configura automáticamente.

Cuando la utilice CloudFormation, podrá reutilizar la plantilla para configurar los HealthImaging recursos de forma coherente y repetida. Describa sus recursos una vez y, a continuación, aprovisiona los mismos recursos una y otra vez en varias Cuentas de AWS regiones.

HealthImaging y CloudFormation plantillas

Para aprovisionar y configurar recursos HealthImaging y servicios relacionados, debe conocer [CloudFormation las plantillas](#). Las plantillas son archivos de texto con formato JSON o YAML. Estas plantillas describen los recursos que desea aprovisionar en sus CloudFormation pilas. Si no estás familiarizado con JSON o YAML, puedes usar CloudFormation Designer para ayudarte a empezar con CloudFormation las plantillas. Para obtener más información, consulte [¿Qué es Designer de CloudFormation ?](#) en la Guía del usuario de AWS CloudFormation .

AWS HealthImaging admite la creación [de almacenes de datos](#) con CloudFormation. Para obtener más información, incluidos ejemplos de plantillas JSON y YAML para el aprovisionamiento de almacenes de HealthImaging datos, consulte la [referencia de tipos de HealthImaging recursos de AWS](#) en la Guía del AWS CloudFormation usuario.

Obtenga más información sobre CloudFormation

Para obtener más información CloudFormation, consulte los siguientes recursos:

- [AWS CloudFormation](#)
- [AWS CloudFormation Guía del usuario](#)
- [Referencia de la API de CloudFormation](#)

- [AWS CloudFormation Guía del usuario de la interfaz de línea de comandos](#)

AWS HealthImaging y puntos finales de VPC de interfaz ()AWS PrivateLink

Puede establecer una conexión privada entre su VPC y crear un punto final AWS HealthImaging de VPC de interfaz. Los puntos de enlace de la interfaz funcionan con una tecnología que puede usar para acceder de forma privada HealthImaging APIs 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 ellas. HealthImaging APIs 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 de conexión 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 de conexión 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 obtener 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*.imagenología médica
- com.amazonaws. *region*.runtime-medical-imaging
- com.amazonaws. *region*.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 asociar una política de punto de conexión con su punto de conexión de VPC que controla 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 conexión de 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 [Creación un rol de IAM para la importación](#).
3. El propietario del almacén de HealthImaging datos debe proporcionar [inputOwnerAccountId](#) del depósito de entrada de Amazon S3 al iniciar el trabajo de importación.

Note

Al proporcionar `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 AWS global se basa en distintas zonas. Regiones de AWS de disponibilidad. Regiones de AWS proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia. 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 uno o varios centros de datos.

Si necesita replicar sus datos o aplicaciones sobre grandes distancias geográficas, utilice las regiones locales de AWS. Una región AWS local es un centro de datos único diseñado para complementar una región existente. AWS. Como todas las Regiones de AWS, las regiones AWS locales están completamente aisladas de las demás Regiones de AWS.

Para obtener más información sobre las zonas de disponibilidad, las Regiones de AWS y las zonas de disponibilidad, consulte [Infraestructura AWS global](#).

Material de HealthImaging referencia de AWS

Note

Todas las acciones y tipos de datos de la HealthImaging API nativa se describen en la [Referencia de HealthImaging API de AWS](#).

Temas

- [Compatibilidad con DICOM para AWS HealthImaging](#)
- [HealthImaging Referencia de AWS](#)

Compatibilidad con DICOM para AWS HealthImaging

AWS HealthImaging admite elementos DICOM y sintaxis de transferencia específicos. Familiarícese con los elementos de datos DICOM 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 las sintaxis HealthImaging de transferencia admitidas y las restricciones de elementos DICOM.

Note

Actualmente, AWS HealthImaging no admite imágenes de segmentación binaria ni datos de píxeles de secuencias de imágenes de iconos.

Temas

- [Clases de SOP compatibles](#)
- [Normalización de metadatos](#)
- [Sintaxis de transferencia compatibles](#)
- [Restricciones de los elementos DICOM](#)
- [Restricciones de los metadatos de DICOM](#)

Clases de SOP compatibles

[Con AWS HealthImaging, puede importar instancias de pares de objetos y servicios \(SOP\) DICOM P10 codificadas con cualquier UID de clase SOP, incluidas las retiradas y las privadas.](#) También se conservan todos los atributos privados.

Normalización de metadatos

Al importar los datos de DICOM P10 a AWS HealthImaging, se transforman en [conjuntos de imágenes](#) compuestos por [metadatos](#) y [marcos de imágenes](#) (datos de píxeles). Durante el proceso de transformación, las claves de HealthImaging metadatos se generan en función de una versión específica del estándar DICOM. HealthImaging actualmente genera y admite claves de metadatos basadas en el diccionario de datos [DICOM PS3 1.6 2022b](#).

AWS HealthImaging admite los siguientes elementos de datos DICOM 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 datos DICOM](#).

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 a nivel de estudio, consulte el [registro de elementos de datos DICOM](#).

AWS HealthImaging admite los siguientes elementos de 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 a nivel de serie, consulte el [registro de elementos de datos DICOM](#).

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 admite la importación de archivos DICOM P10 con diferentes sintaxis de transferencia. Algunos archivos conservan su codificación de sintaxis de transferencia original durante la importación, mientras que la mayoría de los marcos de imagen sin pérdidas se transcodifican durante la importación. El comportamiento de la transcodificación depende de la configuración del almacén de datos. Los almacenes de datos utilizan HTJ2 K como formato de almacenamiento de forma predeterminada, pero se pueden configurar en el momento de la creación para utilizar JPEG 2000 Lossless. En el siguiente ejemplo, se muestra cómo HealthImaging registra a `StoredTransferSyntaxUID` para cada instancia de los [metadatos](#) devueltos por.

[GetImageSetMetadata](#)

```
"Instances": {  
  "999.999.2.19941105.134500.2.101": {  
    "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.90",  
    "ImageFrames": [{ ...
```

Note

Ten en cuenta estos puntos cuando consultes la siguiente tabla:

- Una entrada de UID de sintaxis de transferencia marcada con un asterisco (*) indica que el archivo se ha almacenado en su formato codificado original durante la importación. En el caso de estos archivos, los metadatos `StoredTransferSyntaxUID` ubicados en la instancia coinciden con la sintaxis de transferencia original.
- Una entrada de UID de sintaxis de transferencia marcada sin asterisco indica que un archivo se transcodifica a HTJ2 K sin pérdidas durante la importación y se almacena en. HealthImaging El `StoredTransferSyntaxUID` elemento de los metadatos de la instancia será el formato de almacenamiento del JPEG 2000 de alto rendimiento con las opciones de RPCL: compresión de imagen, solo sin pérdidas (1.2.840.10008.1.2.4.202).

- Si la `StoredTransferSyntaxUID` clave no existe o está configurada en, puede suponer que está codificada en el formato de almacenamiento del almacén de datos configurado. `null`

HealthImaging sintaxis de transferencia compatibles

UID de la sintaxis de transferencia	Nombre de la sintaxis de transferencia
1.2.840.10008.1.2	Endian VR implícita: sintaxis de transferencia predeterminada para DICOM
1.2.840.10008.1.2.1* (la segmentación binaria conserva la codificación original, mientras que la segmentación no binaria se transcodifica a K Lossless RPCL) HTJ2	VR Little Endian explícita
1.2.840.10008.1.2.1.99	VR Little Endian deflated explícita
1.2,840,10008.1.2.2	VR Big Endian explícita
1.2.840.100081.2.4.50*	JPEG Baseline (proceso 1): sintaxis de transferencia predeterminada para la compresión de imágenes JPEG de 8 bits con pérdidas
1.2.840.10008,12,4,51	JPEG Baseline (procesos 2 y 4): sintaxis de transferencia predeterminada para la compresión de imágenes JPEG de 12 bits con pérdidas (solo proceso 4)
1.2,840,10008,12,4,57	JPEG sin pérdidas y no jerárquico (proceso 14)
1.2.840.10008.1.2.4.70	Predicción JPEG sin pérdidas, no jerárquico y de primer orden (Procesos 14 [valor de selección 1]): sintaxis de transferencia predeterminada para la compresión de imágenes JPEG sin pérdidas

UID de la sintaxis de transferencia	Nombre de la sintaxis de transferencia
1.2,840,10008,12,4,80	Compresión de imágenes sin pérdidas JPEG-LS
1.2,840,10008,12,4,81	Compresión de imágenes JPEG-LS con pérdidas (casi sin pérdidas)
1.2,840,10008,12,4,90	Compresión de imágenes JPEG 2000 (solo sin pérdidas)
1.2.840.100081.2.4.91*	Compresión de imágenes JPEG 2000
1.2.840.10008,12,4,92	Compresión de imágenes multicomponente JPEG 2000, parte 2 (solo sin pérdidas)
1.2.840.10008.1.2.4.93	Compresión de imágenes multicomponente JPEG 2000, parte 2
1.2.840.10008.1.2.4.112*	JPEG XL
1.2.840.10008.1.2.4.201	Compresión de imágenes JPEG 2000 de alto rendimiento (solo sin pérdidas)
1.2.840.10008.1.2.4.202	Compresión de imagen JPEG 2000 de alto rendimiento con opciones RPCL (solo sin pérdidas)
1.2.840.10008.1.2.4.203*	Compresión de imágenes JPEG 2000 de alto rendimiento
1.2.840.10008.1.2.5	RLE sin pérdidas
1.2.840.10008.1.2.4.100*, 1.2.840.10008.1.2.4.100,1*	MPEG2 Perfil principal Nivel principal
1.2.840.10008.1.2.4.101*, 1.2.840.10008.1.2.4.101,1*	MPEG2 Perfil principal de alto nivel

UID de la sintaxis de transferencia	Nombre de la sintaxis de transferencia
1.2.840.10008.1.2.4.102*, 1.2.840.10008.1.2.4.102.1*	MPEG-4 AVC/H.264 de perfil alto /nivel 4.1
1.2.840.10008.1.2.4.103*, 1.2.840.10008.1.2.4.103.1*	Perfil alto compatible con MPEG-4 AVC/H.264 BD, nivel 4.1
1.2.840.10008.1.2.4.104*, 1.2.840.10008.1.2.4.104.1*	MPEG-4 AVC/H.264 de perfil alto /nivel 4.2 para vídeo 2D
1.2.840.10008.1.2.4.105*, 1.2.840.10008.1.2.4.105.1*	MPEG-4 AVC/H.264 de alto perfil /nivel 4.2 del vídeo H.264 de la ITU-T
1.2.840.10008.1.2.4.106*, 1.2.840.10008.1.2.4.106.1*	MPEG-4 AVC/H.264 estéreo de perfil alto /nivel 4.2 del vídeo ITU-T H.264
1.2.840.10008.1.2.4.107*	Perfil principal HEVC/H.265 /Vídeo de nivel 5.1
1.2.840.10008.1.2.4.108*	10 perfiles principales HEVC/H.265, nivel 5.1

*Conserva la codificación de la sintaxis de transferencia original durante la importación

Restricciones de los elementos DICOM

Al importar sus datos de imágenes médicas a AWS HealthImaging, se aplican restricciones de longitud máxima a los siguientes elementos DICOM. Para que la importación sea correcta, asegúrese de que sus datos no superen las restricciones de longitud máxima.

Restricciones de los elementos DICOM durante la importación

Palabra clave DICOM	Etiqueta DICOM	Longitud máxima
PatientName	(0010,0010)	256
ID del paciente	(0010,0020)	256
PatientBirthDate	(0010,0030)	18
PatientSex	(0010,0040)	16

Palabra clave DICOM	Etiqueta DICOM	Longitud máxima
StudyInstanceUID	(0020,000D)	256
StudyID	(0020,0010)	256
StudyDescription	(0008,1030)	256
NumberOfStudyRelatedSeries	(0020,1206)	1 000 000
NumberOfStudyRelatedInstances	(0020,1208)	1 000 000
AccessionNumber	(0008,0050)	256
StudyDate	(0008,0020)	18
StudyTime	(0008,0030)	28
SOPInstanceUID	(0008,0018)	256
SeriesInstanceUID	(0020,000E)	256

Restricciones de los metadatos de DICOM

Cuando se utiliza `UpdateImageSetMetadata` para actualizar HealthImaging [los atributos de los metadatos](#), se aplican las siguientes restricciones DICOM.

- No se pueden actualizar ni eliminar los atributos privados de los atributos de Patient/Study/Series/Instance nivel, a menos que la restricción de actualización se aplique tanto a `updateableAttributes` como `removableAttributes`
- No se pueden actualizar los siguientes atributos HealthImaging generados por AWS:
`SchemaVersion`
`DatastoreIdImageSetID`, `PixelData`, `Checksum`, `Width`, `Height`, `MinPixelValue`, `MaxPixelValue`, `FrameSizeInBytes`
- No se pueden actualizar los siguientes atributos DICOM a menos que el force indicador esté activado: `Tag.PixelData`, `Tag.StudyInstanceUID`, `Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID`, `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 estándar DICOM a menos que se active el `force` 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.

HealthImaging Referencia de AWS

Esta sección contiene datos de respaldo relevantes para AWS HealthImaging.

Temas

- [HealthImaging Puntos de enlace y cuotas de AWS](#)
- [Límites de HealthImaging regulación de AWS](#)
- [Verificación de datos de HealthImaging píxeles de AWS](#)
- [HealthImaging Códigos de advertencia](#)
- [Bibliotecas de decodificación de marcos de imágenes para AWS HealthImaging](#)
- [Proyectos de HealthImaging muestra de AWS](#)
- [Uso de este servicio con un AWS SDK](#)

HealthImaging Puntos de enlace y cuotas de AWS

Los siguientes temas contienen información sobre los puntos de enlace y las cuotas de los HealthImaging servicios de AWS.

Temas

- [Puntos de conexión de servicio](#)
- [Cuotas de servicio](#)

Puntos de conexión de servicio

Los puntos de conexión de servicio son URL que identifican un host y un puerto como puntos de entrada de un servicio web. Cada solicitud de servicio web contiene un punto de enlace. La mayoría de AWS los servicios proporcionan puntos de enlace para regiones específicas a fin de permitir una conectividad más rápida. En la siguiente tabla se enumeran los puntos de enlace del servicio de 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	
		medical-imaging-fips.us-east-1.amazonaws.com	HTTPS	
Oeste de EE. UU. (Oregón)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS	
		medical-imaging-fips.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 solicitudes HTTP para llamar a las HealthImaging acciones de AWS, debe utilizar distintos puntos de enlace en función de las acciones a las que se llame. El siguiente menú muestra los puntos de conexión de servicio disponibles para las solicitudes HTTP y las acciones que admiten.

Acciones de la API admitidas para solicitudes HTTP

data store, import, tagging

Se puede acceder a las siguientes acciones de almacenamiento, importación y etiquetado de datos a través del punto final:

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- Iniciar DICOMImport trabajo
- Consigue un DICOMImport trabajo
- Listar DICOMImport trabajos
- 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 de DICOMweb Retrieve WADO-RS. Para obtener más información, consulte [Recuperación de datos DICOM 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

Cuotas de servicio

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 aumento de cuota](#) en la Guía del usuario de Service Quotas.

En la siguiente tabla se enumeran las cuotas predeterminadas de AWS HealthImaging.

Name	Predeterminado	Ajustable	Description (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 DeletedImageSet solicitudes simultáneas por almacén de datos	Cada región admitida: 100	Sí	El número máximo de DeletedImageSet solicitud es 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 Cada una de las demás regiones compatibles: 100	Sí	El número máximo de trabajos de importación simultáneos por banco de datos en la región actual AWS
Máximo de almacenes de datos	Cada región admitida: 10	Sí	El número máximo de almacenes de datos

Name	Predeterminado	Ajuste	Description (Descripción)
			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 de importación DICOM	Cada región admitida: 5000	Sí	El número máximo de archivos en un trabajo de importación de DICOM en la región actual AWS
Número máximo de carpetas anidadas en un trabajo de importación DICOM	Cada región admitida: 10 000	No	El número máximo de carpetas anidadas en un trabajo de importación DICOM 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
El tamaño máximo (en GB) de todos los archivos de un trabajo de importación DICOM	Cada región admitida: 10 gigabytes	No	El tamaño máximo (en GB) de todos los archivos de un trabajo de importación de DICOM en la región actual AWS

Name	Predeterminado	Ajuste	Description (Descripción)
Tamaño máximo (en GB) de cada archivo DICOM P10 de un trabajo de importación DICOM	Cada región admitida: 4 gigabytes	No	El tamaño máximo (en GB) de cada archivo DICOM P10 del trabajo de importación DICOM 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

Límites de HealthImaging regulación de AWS

Su AWS cuenta tiene límites de limitación que se aplican a las acciones de la HealthImaging API de AWS. En todas las acciones se genera un error `ThrottlingException` si se superan los límites de limitación. Para obtener más información, consulte la [referencia de HealthImaging API de AWS](#).

Note

Los límites de limitación se pueden ajustar para todas las acciones de la HealthImaging API. Para solicitar un ajuste del límite de limitación, póngase en contacto con el [Centro de soporte de AWS](#). Para crear un caso, inicia sesión en tu AWS cuenta y selecciona Crear caso.

En la siguiente tabla se enumeran los límites de limitación tanto para [HealthImaging las acciones nativas](#) como para las [representaciones de los DICOMweb servicios](#).

Límites de HealthImaging regulación de AWS

Action	Velocidad de limitación	Ráfaga del limitación
CreateDatastore	0,085 TPS	1 TPS
GetDatastore	10 TPS	20 TPS

Action	Velocidad de limitación	Ráfaga del limitación
ListDatastores	5 TPS	10 TPS
DeleteDatastore	0,085 TPS	1 TPS
Iniciar DICOMImport trabajo	1 TPS	2 consejos
Consigue un DICOMImport trabajo	25 TPS	50 TPS
Listar DICOMImport trabajos	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
Obtenga DICOMInstance *	50 TPS	100 TPS
Obtener DICOMInstance metadatos*	50 TPS	100 TPS
Obtener marcos* DICOMInstance	50 TPS	100 TPS

Action	Velocidad de limitación	Ráfaga del limitación
Obtenga metadatos DICOMSeries	50 TPS	100 TPS

*Representación de un servicio DICOMweb

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 [bibliotecas de decodificación HTJ2 K](#) 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, se genera una suma de verificación de la resolución de fotogramas (IFRC) única e inmutable para cada imagen. CRC32 El valor de la suma de verificación de la IFRC se presenta en el documento de metadatos. `job-output-manifest.json` Para obtener más información, consulte [Introducción a los trabajos de importación](#).
- Una vez importadas las imágenes a un [almacén de HealthImaging datos](#) y transformadas en [conjuntos de imágenes, los fotogramas de imágenes](#) codificadas en HTJ2 K se decodifican inmediatamente y se calculan los nuevos. IFRCs HealthImaging a continuación, compara la resolución completa IFRCs de las imágenes originales con la nueva IFRCs de los marcos de imagen importados para comprobar su precisión.
- En el registro de resultados del trabajo de importación (`job-output-manifest.json`) se captura la correspondiente condición de error descriptivo por imagen 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) de la imagen de resolución completa. Para las imágenes almacenadas con su sintaxis de transferencia original 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 y 1.2.840.10008.1.2.1 (solo segmentación binaria), la suma de verificación se calcula sobre la imagen original. En el caso de las imágenes almacenadas en K HTJ2 Lossless con RPCL, la suma de verificación se calcula sobre la imagen de resolución completa decodificada. Para obtener más información, consulte [Sintaxis de transferencia compatibles](#).

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": "67890678906789012345123451234512",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 512,
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
}]
```

En el caso de las imágenes almacenadas con su sintaxis de transferencia original 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.4.50 y 1.2.840.10008.1.2.1 (solo segmentación binaria), el y no estará disponible. `MinPixelValue` `MaxPixelValue` `FrameSizeInBytes` Indica el tamaño del marco original.

```
"PixelDataChecksumFromBaseToFullResolution": [
  {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": null,
"MaxPixelValue": null,
"FrameSizeInBytes": 429
```

Para las imágenes almacenadas en HTJ2 K Lossless con RPCL, `FrameSizeInBytes` indica el tamaño del marco de la imagen decodificada.

```
"PixelDataChecksumFromBaseToFullResolution": [
  {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": 11,
"MaxPixelValue": 11,
"FrameSizeInBytes": 1652
```

4. En el caso de las instancias que contienen vídeo, HealthImaging realiza una validación ligera del códec que verifica que la sintaxis de transferencia especificada en los metadatos DICOM coincida con el códec de vídeo.

HealthImaging calcula un valor de suma de verificación de la IFRC en el objeto de vídeo original mediante el algoritmo. CRC32 El valor de la suma de verificación de la IFRC se registra en los metadatos y se conserva en `ellosjob-output-manifest.json`. HealthImaging Al igual que ocurre con las imágenes almacenadas en su sintaxis de transferencia original (descrita anteriormente), el `MinPixelValue` y no `MaxPixelValue` estará disponible. `FrameSizeInBytes` Indica el tamaño del marco original.

5. HealthImaging verifique 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 [Bibliotecas de decodificación de marcos de imágenes](#) procesadores que utilizan. HealthImaging Una vez cargada la imagen completa, 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 verificar los datos de píxeles.

HealthImaging Códigos de advertencia

HealthImaging intenta importar todos sus datos de imágenes médicas. Si se detectan datos no conformes o se detectan elementos de datos no reconocidos durante las importaciones, HealthImaging añadirá una de las siguientes advertencias al archivo. `warning.ndjson` También se podrá buscar una advertencia asociada a una instancia importada mediante la acción del `SearchDICOMInstances` elemento. `WarningReason` Las instancias importadas con una advertencia pueden tener un soporte reducido HealthImaging APIs, tal y como se describe a continuación.

HealthImaging Importar códigos de advertencia

Motivo de advertencia (hexadecimal)	Motivo de advertencia (decimal)	Tipo de advertencia (enumeración)	Detalles de la advertencia	Comportamiento resultante
-------------------------------------	---------------------------------	-----------------------------------	----------------------------	---------------------------

Motivos de advertencia según el estándar DICOM

0xB000	45056	COERCIÓN DE ELEMENTOS DE DATOS	La ingesta modificó uno o más elementos de datos durante el almacenamiento de la instancia. Consulte la sección 6.6.1.3 .	n/a
0xB006	45062	ELEMENTOS _DESCARTADOS	La ingesta descartó algunos elementos de datos durante el almacenamiento de la instancia. Consulte la sección 6.6.1.3 .	n/a
0xB007	45063	SOP_CLASS_DATA_MISMATCH	La StoreDICOM acción observó que el conjunto de datos no cumplía con las restricciones de la clase SOP durante el almacenamiento de la instancia.	n/a

Motivos de HealthImaging advertencia de AWS

0xB100	45312	EXCEPCIÓN DE TRANSCODIFICACIÓN	Esta advertencia se produce cuando HealthImaging no se puede transcodificar la instancia PixelData	Si es necesario, es posible que los datos de píxeles se puedan recuperar como un único bloque. Si se
--------	-------	--------------------------------	--	--

Motivo de advertencia (hexadecimal)	Motivo de advertencia (decimal)	Tipo de advertencia (enumeración)	Detalles de la advertencia	Comportamiento resultante
			a HTJ2 K (formato de almacenamiento predeterminado) o si PixelData no se puede transcodificar por otro motivo (por ejemplo, validaciones fallidas, atributos de datos de píxeles incorrectos, etc.). En este caso, los datos de píxeles se almacenarán como un blob.	pasa el comodín «*» como encabezado de Accept, se devolverán transfer-syntax en el formato almacenado.
0xB110	45328	FALLO DE EXTRACCIÓN DE MARCOS	Esta advertencia se produce cuando hay un problema al analizar fotogramas individuales a partir de los PixelData metadatos DICOM proporcionados.	Los datos de píxeles tenían un formato incorrecto y no se pueden recuperar; utilícelos GetDICOMInstance para recuperar la instancia completa
0xB111	45329	DISCORDANCIA DEL NÚMERO_MARCOS	Esta advertencia se produce cuando el elemento DICOM NumberOfFrames «» no coincide con el número real de «fragmentos» de imagen del archivo DICOM de entrada.	Los datos de píxeles tenían un formato incorrecto y no se pueden recuperar; utilícelos GetDICOMInstance para recuperar la instancia completa

Motivo de advertencia (hexadecimal)	Motivo de advertencia (decimal)	Tipo de advertencia (enumeración)	Detalles de la advertencia	Comportamiento resultante
0xB112	45330	TABLA_OFFSET NO VÁLIDA	Esta advertencia se produce cuando la tabla de desplazamiento de los fragmentos del archivo DICOM de entrada no coincide con la longitud real del fotograma y podría dar lugar a fotogramas con un formato incorrecto según su gravedad.	Los datos de píxeles tenían un formato incorrecto y no se pueden recuperar; utilícelos <code>GetDICOMInstance</code> para recuperar la instancia completa
0xB120	45344	SINTAXIS DE TRANSFERENCIA NO COMPATIBLE	Esta advertencia se produce cuando HealthImaging encuentra una sintaxis de transferencia no reconocida o no admitida. Cuando esto suceda, HealthImaging almacenará los datos de píxeles como una mancha.	Si es necesario, es posible que los datos de píxeles se puedan recuperar como un único bloque. Si se coloca el comodín «*» <code>transfer-syntax</code> en el encabezado de <code>Accept</code> , se devolverán en el formato almacenado.
0xB201	45570	FORMATO_UID_VÁLIDO	Esta advertencia se produce cuando uno o más elementos del UID infringen la representación del valor DICOM (por ejemplo) 1.2.3..4	n/a

Motivo de advertencia (hexadecimal)	Motivo de advertencia (decimal)	Tipo de advertencia (enumeración)	Detalles de la advertencia	Comportamiento resultante
0xB202	45571	DICOM_VALOR_LONGITUD NO VÁLIDO	Esta advertencia se produce cuando un elemento DICOM tiene una longitud superior a la admitida por la representación de valores DICOM, lo que podría introducir comportamientos no válidos en las acciones. search/retrieve	Es posible que algunos campos no se puedan analizar y, por lo tanto, no se puedan buscar en ellos (es decir, o) StudyDate StudyTime
0xBFFF	47513	OTHER	Esta advertencia se produce cuando hay una advertencia no detectada que HealthImaging no se captura como un código de advertencia específico.	Los datos de píxeles tenían un formato incorrecto y no se pueden recuperar; utilícelos GetDICOMInstance para recuperar la instancia completa

Bibliotecas de decodificación de marcos de imágenes para AWS HealthImaging

Durante la [importación](#), algunas sintaxis de transferencia conservan su codificación original, mientras que otras se transcodifican a JPEG 2000 (HTJ2K) sin pérdidas de alto rendimiento de forma predeterminada o a JPEG 2000 Lossless (cuando está configurada), según la configuración del almacén de datos. HTJ2K ofrece una visualización de imágenes rápida y uniforme y un acceso universal a las funciones avanzadas de K. HTJ2 Como los marcos de imagen se codifican en HTJ2 K o JPEG 2000 Lossless durante la importación, deben decodificarse antes de visualizarlos en un visor de imágenes. Para obtener información sobre cómo determinar las sintaxis de transferencia, consulte

Sintaxis de transferencia compatibles. Para obtener información sobre cómo determinar las sintaxis de transferencia, consulte. [Sintaxis de transferencia compatibles](#)

 Note

HTJ2K se define en la [parte 15 de la norma JPEG2 000 \(ISO/IEC 15444-15:2019\)](#). HTJ2K conserva las funciones avanzadas de JPEG2 000, como la escalabilidad de la resolución, los recintos, el ordenamiento en teselas, la alta profundidad de bits, los múltiples canales y la compatibilidad con espacios de color.

Temas

- [Bibliotecas de decodificación de cuadros de imágenes](#)
- [Visores de imágenes](#)

Bibliotecas de decodificación de cuadros de imágenes

Según el lenguaje de programación que utilice, le recomendamos las siguientes bibliotecas de decodificación para decodificar los [marcos de imagen](#).

- [NVIDIA nv JPEG2 000: 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](#)

Proyectos de HealthImaging muestra de AWS

AWS HealthImaging ofrece los siguientes proyectos de ejemplo en GitHub.

[Visor OHIF integrado a AWS HealthImaging mediante OIDC](#)

[Este AWS Cloud Development Kit \(AWS CDK\) proyecto implementa el visor OHIF en Amazon. CloudFront](#) El visor está integrado en un almacén de datos de Amazon Web Services como fuente de DICOMWeb datos y con [Amazon](#) Cognito como proveedor de identidad para la autenticación mediante OIDC.

[Ingestión de DICOM de las instalaciones a AWS HealthImaging](#)

Un proyecto AWS sin servidor para implementar una solución perimetral de IoT que reciba archivos DICOM de una fuente DICOM DIMSE (PACS, VNA, escáner CT) y los almacene en un bucket seguro de Amazon S3. La solución indexa los archivos DICOM de una base de datos y pone en cola cada serie DICOM para importarla en AWS. HealthImaging Se compone de un componente que se ejecuta en la periferia y es administrado por [AWS IoT Greengrass](#) una canalización de entrada de DICOM que se ejecuta en la nube. AWS

[Proxy Tile Level Marker \(TLM\)](#)

Un [AWS Cloud Development Kit \(AWS CDK\)](#) proyecto para recuperar marcos de imágenes de AWS HealthImaging mediante marcadores de nivel de tesela (TLM), una función del JPEG 2000 (K) de alto rendimiento. HTJ2 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 punto final HTTPS que almacene en caché (mediante GET) y entregue marcos de imágenes desde el borde. Por defecto, el punto de conexión autentica las solicitudes con un token web JSON (JWT) de Amazon 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.

[Interfaz de usuario de AWS HealthImaging Viewer](#)

Un [AWS Amplify](#) proyecto para implementar una interfaz de usuario frontend con autenticación de backend con la que puede ver los atributos de metadatos del conjunto de imágenes y los marcos

de imágenes (datos de píxeles) almacenados en AWS HealthImaging mediante la decodificación progresiva. De forma opcional, puedes integrar los proyectos de and/or Amazon CloudFront Delivery proxy Tile Level Marker (TLM) anteriores para cargar marcos de imágenes mediante un método alternativo.

[HealthImaging DICOMweb Proxy de AWS](#)

Un proyecto basado en Python para habilitar los terminales 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 [AWS HealthImaging Samples](#) en GitHub.

Uso de este servicio con un AWS SDK

AWS Los kits de desarrollo de software (SDKs) 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 para C++	AWS SDK para C++ ejemplos de código
AWS CLI	AWS CLI ejemplos de código
AWS SDK para Go	AWS SDK para Go ejemplos de código
AWS SDK para Java	AWS SDK para Java ejemplos de código
AWS SDK para JavaScript	AWS SDK para JavaScript ejemplos de código
AWS SDK para Kotlin	AWS SDK para Kotlin ejemplos de código
AWS SDK para .NET	AWS SDK para .NET ejemplos de código

Documentación de SDK	Ejemplos de código
AWS SDK para PHP	AWS SDK para PHP ejemplos de código
Herramientas de AWS para PowerShell	Herramientas de AWS para PowerShell ejemplos de código
AWS SDK para Python (Boto3)	AWS SDK para Python (Boto3) ejemplos de código
AWS SDK para Ruby	AWS SDK para 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.

Optimización de costes

HealthImaging está diseñado para optimizar los costos de almacenamiento al mover automáticamente los datos al nivel de acceso más rentable cuando cambian los patrones de acceso. A medida que los patrones de uso cambian con el tiempo, la organización inteligente en niveles proporciona una gestión automática del ciclo de vida de los datos DICOM sin sobrecargas operativas. HealthImaging tiene dos niveles de almacenamiento:

- Almacenamiento en niveles de acceso frecuente para datos DICOM recién importados o a los que se accede con regularidad.
- Archive el almacenamiento en niveles de acceso instantáneo para los datos DICOM a los que no se haya accedido recientemente. Reduce los costos de almacenamiento para el archivado a largo plazo y, al mismo tiempo, mantiene las latencias de acceso de milisegundos.

Se le facturará por el tamaño total de almacenamiento de todos los conjuntos de imágenes de un almacén de datos. Ambos niveles de almacenamiento se facturan por GB al mes y no hay ninguna tarifa por conjunto de imágenes. Tampoco se cobran cargos de recuperación por mover datos entre niveles de almacenamiento.

Note

Los conjuntos de imágenes se facturan con un tamaño mínimo de 5 MB. HealthImaging acepta conjuntos de imágenes de menos de 5 MB, pero estos objetos más pequeños tienen una tarifa de 5 MB.

Cómo funciona la organización inteligente en niveles

Sus datos se almacenan en el nivel de acceso frecuente cuando se crean nuevos conjuntos de imágenes. Los conjuntos de imágenes se pueden crear importando nuevos datos (mediante Import Jobs o DICOMweb STOW-RS) o invocándolos. CopyImageSet HealthImaging mueve automáticamente los conjuntos de imágenes a los que no se ha accedido durante 30 días consecutivos al nivel Archive Instant Access, y los conjuntos de imágenes permanecen en el nivel Archive Instant Access hasta que se vuelva a acceder a ellos.

Las siguientes acciones constituyen el acceso a los datos DICOM que mueven automáticamente los conjuntos de imágenes del nivel Archive Instant Access al nivel de acceso frecuente:

- Invocar [GetImageSetMetadataGetImageFrame](#), o cualquier acción de [DICOMweb WADO-RS](#).
- [CopyImageSetUpdateImageSetMetadata](#) Invocando o. En el caso de las operaciones de copia, solo los conjuntos de imágenes replicadas se agrupan en niveles para el acceso frecuente. En el caso de Copiar con destino, el conjunto de imágenes de destino está agrupado en niveles.
- Visualización y descarga de los datos del conjunto de imágenes a través de la consola AWS HealthImaging de administración.

Las acciones anteriores permiten que los conjuntos de imágenes pasen al nivel de acceso frecuente e impiden que los conjuntos de imágenes del nivel de acceso frecuente pasen al nivel de acceso instantáneo a archivos durante otros 30 días. El acceso a los conjuntos de imágenes se puede realizar a través AWS de la consola de administración o mediante interfaces programáticas, como la AWS CLI o. AWS SDKs Otras acciones no constituyen acceso y, por lo tanto, no moverán automáticamente los objetos del nivel Archive Instant Access al nivel de acceso frecuente. La siguiente es una muestra, no una lista definitiva, de dichas acciones:

- Acciones en los almacenes de datos ([CreateDatastore](#) y [GetDatastore](#))
- Acciones de [lista](#) ([Listar DICOMImport](#) [ListImageSetVersionstrabajos](#) y [ListTagsForResource](#))
- Acciones de búsqueda ([SearchImageSets](#) y consultas [DICOMweb QIDO-RS](#))
- Invocando, [GetImageSet](#) o [TagResourceUntagResource](#)
- Eliminar operaciones

Los datos importados a HealthImaging se cobran por un período mínimo de almacenamiento de 30 días. Puedes eliminar tus datos en cualquier momento, pero se cobrará el resto del tiempo mínimo por cada imagen eliminada antes de 30 días desde la importación.

Estimación del almacenamiento de datos estructurados

HealthImaging almacena parte de la información del encabezado DICOM en un almacenamiento indexado. Se le cobra por este consumo de almacenamiento estructurado independientemente del nivel de almacenamiento del conjunto de imágenes, y estos cargos son adicionales. Puede estimar el consumo de almacenamiento estructurado multiplicando la cantidad de recursos DICOM en sus almacenes de datos por los tamaños de los registros indexados por recurso. Los siguientes valores son aproximados.

Recurso DICOM	Tamaños indexados (bytes)
Estudio	1024
Serie	830
Instancia	680

HealthImaging Versiones de AWS

En la siguiente tabla se muestra cuándo se publicaron las funciones y actualizaciones para el HealthImaging servicio y la documentación de AWS. Para más información sobre una versión, consulte el tema enlazado.

Cambio	Descripción	Fecha
Compatibilidad con JPEG XL para almacenes de HealthImaging datos de AWS	HealthImaging admite la importación y el almacenamiento de archivos DICOM con pérdidas en la sintaxis de transferencia JPEG XL (1.2.840.10008.1.2.4.112). Para obtener más información , consulte Sintaxis de transferencia compatibles.	2 de febrero de 2026
Importación DICOM mejorada con soporte de advertencia para datos no conformes	HealthImaging ahora importa datos DICOM rechazados anteriormente, aceptando archivos no conformes a la vez que proporciona advertencias detalladas a través de eventos. EventBridge Los trabajos de importación ahora generan una carpeta de ADVERTENCIA que contiene warning.ndjson archivos con códigos de motivo de advertencia específicos para los datos no conformes. La actualización añade compatibilidad con elementos DICOM con capacidad de búsqueda WarningReason (0008,1196)	8 de diciembre de 2025

e introduce un `transfer-syntax=*` parámetro para recuperar las instancias en su formato almacenado cuando no es posible transcodificarlas.

[Consulte Descripción de los trabajos de importación y los códigos de advertencia para obtener más información. HealthImaging](#)

[Compatibilidad con JPEG 2000 Lossless para los almacenes de datos de AWS HealthImaging](#)

AWS HealthImaging ahora admite la codificación nativa JPEG 2000 Lossless para imágenes médicas, lo que le permite crear almacenes de datos que persisten y recuperar fotogramas de imágenes sin pérdidas en formato JPEG 2000 sin necesidad de transcodificarlos. [Esto permite una recuperación con menor latencia para las aplicaciones que requieren el formato JPEG 2000 Lossless cuando se especifica al crear un almacén de datos.](#)
`-lossless-storage-format JPEG_2000_LOSSLESS`

25 de noviembre de 2025

[Diseño de mejoras de búsqueda del QIDO-RS](#)

HealthImaging ahora admite la búsqueda con caracteres comodín en los principales atributos del DICOM (nombre del paciente, nombre del médico remitente, identificación del paciente, descripción del estudio, modalidad y número de acceso) y funciones de búsqueda difusa de los nombres de los Patient/Referring médicos para incluir términos incompletos o mal escritos. La actualización también amplía las funciones de consulta de la API QIDO-RS para incluir búsquedas por fecha de nacimiento de los pacientes y descripciones de los estudios, lo que mejora la capacidad de localizar los estudios y series relevantes. Consulte [Búsqueda de datos DICOM en HealthImaging para obtener más información.](#)

15 de septiembre de 2025

[Autorización de OpenID Connect \(OIDC\) para DICOMweb APIs](#)

HealthImaging ahora admite la autorización de token portador de OpenID Connect (OIDC) en todos los sistemas. DICOMweb APIs Esto añade una interoperabilidad OAuth 2.0 basada en estándares con visores y kits de herramientas comunes (por ejemplo, OHIF, SLIM, MONAI) al aceptar IDP emitido en el encabezado. JWTs Authorization [Consulte DICOMweb APIs la](#) autenticación OIDC para obtener más información.

3 de septiembre de 2025

[Support para importación DICOMweb de datos y datos DICOMweb masivos](#)

HealthImaging HealthImaging admite el almacenamiento de archivos DICOM P10 mediante el protocolo STOW-RS. DICOMweb Consulte [Importación de datos para obtener más información.](#) Además, HealthImaging es compatible con DICOM Bulkdata para garantizar una recuperación de metadatos uniforme y de baja latencia. Consulte [Obtener datos masivos de DICOM para obtener más información.](#)

30 de junio de 2025

[Organización automática de datos, búsqueda con DICOMweb QIDO-RS y mejoras en WADO-RS DICOMweb](#)

HealthImaging proporciona una organización automática de los datos del DICOM P10 al importarlos según la jerarquía a nivel de paciente, estudio y serie del estándar DICOM. Para obtener más información, consulte [Descripción](#) de los trabajos de importación. HealthImaging admite búsquedas enriquecidas, según el estándar DICOMweb QIDO-RS. Consulte [Búsqueda de datos DICOM en HealthImaging](#) para obtener más información. HealthImaging permite recuperar los metadatos de todas las instancias DICOM de una serie mediante una sola acción de API, tal y como se describe en [Obtener metadatos de series DICOM](#) de HealthImaging.

22 de mayo de 2025

[La creación de conjuntos de imágenes utiliza menos elementos DICOM](#)

HealthImaging reduce el número de elementos que se utilizan al agrupar los objetos DICOM P10 entrantes en conjuntos de imágenes. Para obtener más información, consulte [¿Qué es un conjunto de imágenes?](#).

27 de enero de 2025

[Soporte con pérdida para Start Job DICOMImport](#)

HealthImaging permite importar y almacenar archivos DICOM con pérdidas (1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50) y archivos de segmentación binaria en su formato original. Para obtener [más](#) información, consulte [Sintaxis de transferencia compatibles](#).

1 de noviembre de 2024

[Soporte de recuperación con pérdidas DICOMweb APIs](#)

HealthImaging permite recuperar imágenes e instancias almacenadas en 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50 y 1.2.840.10008.1.2.1 (solo segmentación binaria) en su formato original o en Explicit VR Little Endian (1.2.840.10008.1.2.1). Para obtener [más información, consulte Sintaxis de transferencia compatibles y recuperación de datos](#) DICOM.

1 de noviembre de 2024

[Importaciones más rápidas para patología digital](#)

HealthImaging admite trabajos de importación de patología digital DICOM (WSI) hasta 6 veces más rápidos.

1 de noviembre de 2024

[Soporte de segmentación binaria](#)

HealthImaging admite tanto la ingesta como la recuperación de archivos de segmentación binaria DICOM. [Para obtener más información, consulte Sintaxis de transferencia compatibles.](#)

1 de noviembre de 2024

[Vuelva 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 referencia de la HealthImaging API de AWS.

24 de julio de 2024

[Forzar la funcionalidad para la modificación de conjuntos de imágenes](#)

24 de julio de 2024

HealthImaging proporciona el tipo de Overrides datos con el parámetro de forced solicitud opcional. La configuración de este parámetro fuerza las CopyImageSet acciones UpdateImageSetMetadata y, incluso si los metadatos a nivel de paciente, estudio o serie no coinciden. Para obtener más información, consulte [Anulaciones](#) en la referencia de HealthImaging API de AWS.

- UpdateImageSetMetadata funcionalidad de fuerza: HealthImaging introduce el parámetro de force solicitud opcional para actualizar los siguientes atributos:
 - Tag.StudyInstanceUID , Tag.SeriesInstanceUID , Tag.SOPInstanceUID , y Tag.StudyID
 - Añadir, eliminar o actualizar elementos de datos DICOM privados a nivel de instancia

Para obtener más información, consulte [UpdateImageSetMetadata](#) la referencia

de la HealthImaging API de AWS.

- `CopyImageSet` funcionalidad de fuerza: HealthImaging introduce el parámetro de `force` solicitud opcional para copiar conjuntos de imágenes. La configuración de este parámetro fuerza la `CopyImageSet` acción, incluso si los metadatos a nivel de paciente, estudio o serie no coinciden entre `sourceImageSet` y `destinationImageSet`. En estos casos, los metadatos incoherentes permanecen inalterados en el `destinationImageSet`. Para obtener más información, consulte [CopyImageSet](#) la referencia de la HealthImaging API de AWS.

[Copie subconjuntos de instancias de SOP](#)

HealthImaging mejora la `CopyImageSet` acción para que pueda elegir una o más instancias de SOP de una `sourceImageSet` para copiar a una `destinationImageSet`. Para obtener más información, consulte [Copiar un conjunto de imágenes](#).

24 de julio de 2024

[GetDICOMInstanceMetadata](#) para devolver los metadatos de una instancia DICOM

HealthImaging proporciona la `GetDICOMInstanceMetadata` API para devolver los metadatos de la parte 10 de DICOM (.jsonarchivo). Para obtener más información, consulte [Obtener metadatos de instancias](#).

11 de julio de 2024

[GetDICOMInstanceFrames](#) para devolver marcos de instancias DICOM (datos de píxeles)

HealthImaging proporciona la `GetDICOMInstanceFrames` API para devolver los fotogramas de la parte 10 de DICOM (multipart solicitud). Para obtener más información, consulte [Obtener marcos de instancias](#).

11 de julio de 2024

[Soporte mejorado para la importación de datos DICOM no estándar](#)

28 de junio de 2024

HealthImaging proporciona soporte para la importación de datos que incluyen desviaciones del estándar DICOM. Para obtener más información, consulte [Restricciones de elementos DICOM](#).

- Los siguientes elementos de datos DICOM 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 UID, Treatment Session UID, Manufacturer's Device Class UID, Device UID, y Acquisition UID :
 - El primer elemento de cualquier UID puede ser cero
 - UIDs puede empezar con uno o más ceros a la izquierda

- UIDs puede tener una longitud máxima de 256 caracteres

[Notificaciones de eventos](#)

HealthImaging se integra con Amazon EventBridge para admitir aplicaciones basadas en eventos. [Para obtener más información, consulte Uso. EventBridge](#)

5 de junio de 2024

[GetDICOMInstance para devolver los datos de la instancia DICOM](#)

HealthImaging proporciona el GetDICOMInstance servicio de devolución de datos de instancia (.dcmarchivo) de la parte 10 de DICOM. Para obtener más información, consulte [Obtener una instancia](#).

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](#).

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](#).

3 de abril de 2024

- 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
- Los parámetros de la serie DICOM se devuelven en las respuestas

[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 de JPEG Lossless y K HTJ2](#)

HealthImaging proporciona compatibilidad con las siguientes sintaxis de transferencia para la importación de trabajos. Para obtener más información, consulte [Sintaxis de transferencia compatibles](#).

16 de febrero de 2024

- 1.2.840.10008.1.2.4.57 — JPEG sin pérdidas y no jerárquico (proceso 14)
- 1.2.840.10008.1.2.4.201: compresión de imágenes JPEG 2000 de alto rendimiento (solo sin pérdidas)
- 1.2.840.10008.1.2.4.202: compresión de imágenes JPEG 2000 de alto rendimiento con opciones RPCL (solo sin pérdidas)
- 1.2.840.10008.1.2.4.203: compresión de imágenes JPEG 2000 de alto rendimiento

[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](#).

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 [Cuotas de servicio](#).

1 de diciembre de 2023

[Importaciones más rápidas](#)

HealthImaging proporciona importaciones 20 veces más rápidas en todas las regiones compatibles. Para obtener más información, consulte Puntos [finales del servicio](#).

1 de diciembre de 2023

[CloudFormation soporte](#)

HealthImaging admite la infraestructura como código (IaC) para el aprovisionamiento de almacenes de datos. Para obtener más información, consulte [Crear HealthImaging recursos](#) con 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). Para obtener más información, consulte los [puntos de enlace del servicio](#).

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.