



Developer Guide

AWS HealthImaging



AWS HealthImaging: Developer Guide

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS HealthImaging?	1
Important notice	2
Features	2
Related services	3
Accessing	4
HIPAA	5
Pricing	5
Getting started	6
Concepts	6
Data store	6
Image set	7
Metadata	7
Image frame	7
Setting up	7
Sign up for an AWS account	8
Create a user with administrative access	8
Create S3 buckets	10
Create a data store	10
Create an IAM user	11
Create an IAM role	11
Install the AWS CLI	14
Tutorial	14
Managing data stores	16
Creating a data store	16
Getting data store properties	23
Listing data stores	29
Deleting a data store	36
Understanding storage tiers	42
Importing imaging data	45
Understanding import jobs	45
Starting an import job	48
Getting import job properties	56
Listing import jobs	62
Accessing image sets	68

Understanding image sets	68
Searching image sets	74
Getting image set properties	99
Getting image set metadata	104
Getting image set pixel data	113
Modifying image sets	121
Listing image set versions	121
Updating image set metadata	127
Copying an image set	145
Deleting an image set	160
Tagging resources	166
Tagging a resource	166
Listing tags for a resource	171
Untagging a resource	175
Code examples	181
Basics	187
Hello HealthImaging	188
Actions	194
Scenarios	328
Get started with image sets and image frames	328
Tagging a data store	383
Tagging an image set	393
DICOMweb	404
Retrieving data	404
Getting an instance	405
Getting instance metadata	407
Getting instance frames	409
Monitoring	412
Using CloudTrail	412
Creating a trail	413
Understanding log entries	414
Using CloudWatch	416
Viewing HealthImaging metrics	417
Creating an alarm	417
Using EventBridge	417
HealthImaging events sent to EventBridge	418

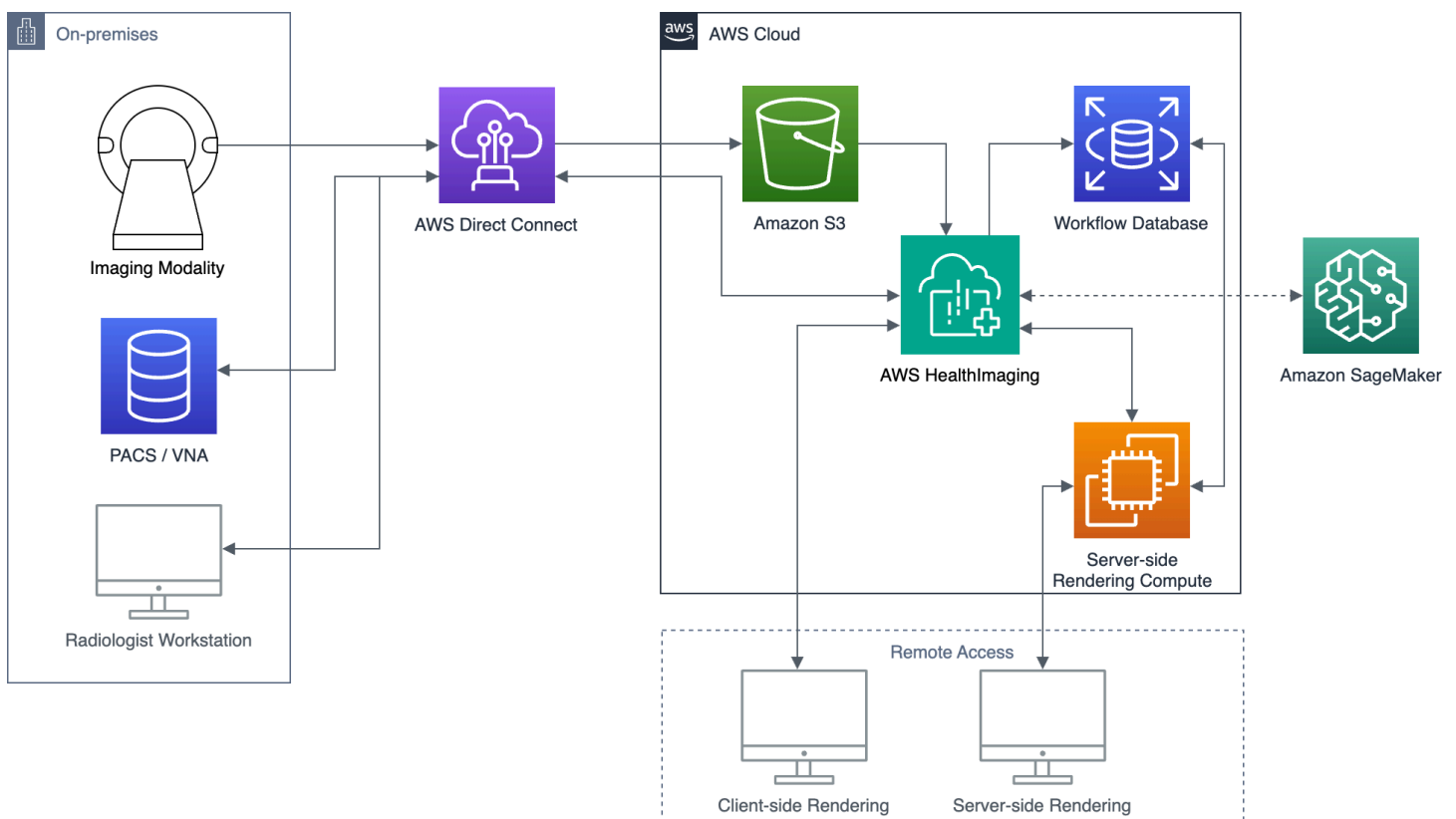
HealthImaging event structure and examples	419
Security	434
Data protection	435
Data encryption	436
Network traffic privacy	445
Identity and Access Management	445
Audience	446
Authenticating with identities	446
Managing access using policies	450
How AWS HealthImaging works with IAM	452
Identity-based policy examples	459
AWS managed policies	462
Troubleshooting	464
Compliance validation	466
Infrastructure security	467
Infrastructure as code	467
HealthImaging and AWS CloudFormation templates	468
Learn more about AWS CloudFormation	468
VPC endpoints	468
Considerations for VPC endpoints	469
Creating a VPC endpoint	469
Creating a VPC endpoint policy	470
Cross-account import	471
Resilience	473
Reference	474
DICOM support	474
Supported SOP classes	475
Metadata normalization	475
Supported transfer syntaxes	480
DICOM element constraints	482
DICOM metadata constraints	483
Pixel data verification	484
HTJ2K decoding libraries	486
HTJ2K decoding libraries	486
Image viewers	487
Endpoints and quotas	487

Service endpoints	487
Service quotas	490
Throttling limits	492
Sample projects	494
Working with AWS SDKs	495
Releases	497

What is AWS HealthImaging?

AWS HealthImaging is a HIPAA eligible service that empowers health care providers, life science organizations, and their software partners to store, analyze, and share medical images in the cloud at petabyte scale. HealthImaging use cases include:

- **Enterprise imaging** – Store and stream your medical imaging data directly from AWS Cloud while preserving low-latency performance and high availability.
- **Long-term image archival** – Save cost on long-term image archival while maintaining subsecond image retrieval access.
- **AI/ML development** – Run artificial intelligence and machine learning (AI/ML) inference over your imaging archive with support from other tools and services.
- **Multimodal analysis** – Combine your clinical imaging data with AWS HealthLake (health data) and AWS HealthOmics (omics data) to deliver insights for precision medicine.



AWS HealthImaging provides access to image data (e.g. X-Ray, CT, MRI, Ultrasound) so that medical imaging applications built in the cloud can achieve performance previously only possible on-

premises. With HealthImaging, you reduce infrastructure costs by running your medical imaging applications at scale from a single, authoritative copy of each medical image in AWS Cloud.

Topics

- [Important notice](#)
- [Features of AWS HealthImaging](#)
- [Related AWS services](#)
- [Accessing AWS HealthImaging](#)
- [HIPAA eligibility and data security](#)
- [Pricing](#)

Important notice

AWS HealthImaging is not a substitute for professional medical advice, diagnosis, or treatment, and is not intended to cure, treat, mitigate, prevent, or diagnose any disease or health condition. You are responsible for instituting human review as part of any use of AWS HealthImaging, including in association with any third-party product intended to inform clinical decision-making. **AWS HealthImaging should only be used in patient care or clinical scenarios after review by trained medical professionals applying sound medical judgment.**

Features of AWS HealthImaging

AWS HealthImaging provides the following features.

Developer-friendly DICOM metadata

AWS HealthImaging simplifies application development by returning DICOM metadata in a developer-friendly format. After importing your imaging data, individual metadata attributes are accessible using human-friendly keywords rather than unfamiliar group/element hexadecimal numbers. Patient, Study, and Series level DICOM elements are [normalized](#), eliminating the need for application developers to deal with inconsistencies between SOP Instances. In addition, metadata attribute values are directly accessible in native runtime types.

SIMD-accelerated image decoding

AWS HealthImaging returns image frames (pixel data) encoded as High Throughput JPEG 2000 (HTJ2K) images, an advanced image compression codec. HTJ2K takes advantage of single

instruction multiple data (SIMD) on modern processors to deliver new levels of performance. HTJ2K is an order of magnitude faster than JPEG2000 and at least twice as fast as all other DICOM transfer syntaxes. WASM-SIMD can be utilized to bring this extreme speed to zero footprint web viewers. For more information, see [Supported transfer syntaxes](#).

Pixel data verification

AWS HealthImaging provides built-in pixel data verification by checking the lossless encoding and decoding state of every image during import. For more information, see [Pixel data verification](#).

Industry-leading performance

AWS HealthImaging sets a new standard for image loading performance thanks to its efficient metadata encoding, lossless compression, and progressive resolution data access. Efficient metadata encoding enables image viewers and AI algorithms to understand the contents of a DICOM study without having to load the image data. Images load faster without any compromise in image quality thanks to advanced image compression. Progressive resolution enables even faster image loading for thumbnails, regions of interest, and low-resolution mobile devices.

Scalable DICOM imports

AWS HealthImaging imports leverage modern cloud native technologies to import multiple DICOM studies in parallel. Historical archives can be imported quickly without impacting clinical workloads for new data. For information about supported SOP instances and transfer syntaxes, see [DICOM support](#).

Related AWS services

AWS HealthImaging features tight integration with other AWS services. A knowledge of the following services is useful to fully leverage HealthImaging.

- [AWS Identity and Access Management](#) – Use IAM to securely manage identities and access to HealthImaging resources.
- [Amazon Simple Storage Service](#) – Use Amazon S3 as a staging area to import DICOM data into HealthImaging.
- [Amazon CloudWatch](#) – Use CloudWatch to observe and monitor HealthImaging resources.
- [AWS CloudTrail](#) – Use CloudTrail to track HealthImaging user activity and API usage.

- [AWS CloudFormation](#) – Use AWS CloudFormation to implement infrastructure as code (IaC) templates to create resources in HealthImaging.
- [AWS PrivateLink](#) – Use Amazon VPC to establish connectivity between HealthImaging and [Amazon Virtual Private Cloud](#) without exposing data to the internet.
- [Amazon EventBridge](#) – Use EventBridge to create scalable, event-driven applications by creating rules that route HealthImaging events to targets.

Accessing AWS HealthImaging

You can access AWS HealthImaging using the AWS Management Console, AWS Command Line Interface and the AWS SDKs. This guide provides procedural instructions for the AWS Management Console and code examples for the AWS CLI and AWS SDKs.

AWS Management Console

The AWS Management Console provides a web-based user interface for managing HealthImaging and its associated resources. If you've signed up for an AWS account, you can sign in to the [HealthImaging console](#).

AWS Command Line Interface (AWS CLI)

The AWS CLI provides commands for a broad set of AWS products, and is supported on Windows, Mac, and Linux. For more information, see the [AWS Command Line Interface User Guide](#).

AWS SDKs

AWS SDKs provide libraries, code examples, and other resources for software developers. These libraries provide basic functions that automate tasks such as cryptographically signing your requests, retrying requests, and handling error responses. For more information, see [Tools to Build on AWS](#).

HTTP requests

You can call HealthImaging actions using HTTP requests, but you must specify different endpoints depending on the type of actions being used. For more information, see [Supported API actions for HTTP requests](#).

HIPAA eligibility and data security

This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see [HIPAA Overview](#).

Connections to HealthImaging containing PHI and personally identifiable information (PII) must be encrypted. By default, all connections to HealthImaging use HTTPS over TLS. HealthImaging stores encrypted customer content and operates according to the [AWS Shared Responsibility Model](#).

For information about compliance, see [Compliance validation for AWS HealthImaging](#).

Pricing

HealthImaging helps you automate the lifecycle management of clinical data with intelligent tiering. For more information, see [Understanding storage tiers](#).

For general pricing information, see [AWS HealthImaging pricing](#). To estimate costs, use the [AWS HealthImaging pricing calculator](#).

Getting started with AWS HealthImaging

To start using AWS HealthImaging, set up an AWS account and create an AWS Identity and Access Management user. To use the [AWS CLI](#) or the [AWS SDKs](#), you must install and configure them.

After learning about HealthImaging concepts and setting up, a short tutorial with code examples is available to help get you started.

Topics

- [AWS HealthImaging concepts](#)
- [Setting up AWS HealthImaging](#)
- [AWS HealthImaging tutorial](#)

AWS HealthImaging concepts

The following terminology and concepts are central to your understanding and use of AWS HealthImaging.

Concepts

- [Data store](#)
- [Image set](#)
- [Metadata](#)
- [Image frame](#)

Data store

A data store is a repository of medical imaging data that resides within a single AWS Region. An AWS account can have zero or many data stores. A data store has its own AWS KMS encryption key, so data in one data store can be physically and logically isolated from data in other data stores. Data stores support access control using IAM roles, permissions, and attribute-based access control.

For more information, see [Managing data stores](#) and [Understanding storage tiers](#).

Image set

An image set is an AWS concept that defines an abstract grouping mechanism for optimizing related medical imaging data. When you import your DICOM P10 imaging data into an AWS HealthImaging data store, it is transformed into image sets comprised of [metadata](#) and [image frames](#) (pixel data). Importing DICOM P10 data results in image sets that contain DICOM metadata and image frames for one or more Service-Object Pair (SOP) instances in the same DICOM Series.

For more information, see [Importing imaging data](#) and [Understanding image sets](#).

Metadata

Metadata is the non-pixel attributes that exist within an [image set](#). For DICOM, this includes patient demographics, procedure details, and other acquisition-specific parameters. AWS HealthImaging separates the image set into metadata and image frames (pixel data) so applications can access it quickly. This is helpful for image viewers, analytics, and AI/ML use cases that don't require pixel data. DICOM data [normalizes](#) at the Patient, Study, and Series levels, eliminating inconsistencies. This simplifies use of the data, increases safety, and improves access performance.

For more information, see [Getting image set metadata](#) and [Metadata normalization](#).

Image frame

An image frame is the pixel data that exists within an [image set](#) to make up a 2D medical image. Some files retain their original transfer syntax encoding during import, while others are transcoded to High-Throughput JPEG 2000 (HTJ2K) lossless by default. If an image frame is encoded in HTJ2K, it must be decoded prior to viewing in an image viewer. For more information, see [Supported transfer syntaxes](#), [Getting image set pixel data](#), and [HTJ2K decoding libraries](#).

Setting up AWS HealthImaging

You must set up your AWS environment before using AWS HealthImaging. The following topics are prerequisites for the [tutorial](#) located in the next section.

Topics

- [Sign up for an AWS account](#)
- [Create a user with administrative access](#)
- [Create S3 buckets](#)

- [Create a data store](#)
- [Create an IAM user with HealthImaging full access permission](#)
- [Create an IAM role for import](#)
- [Install the AWS CLI \(optional\)](#)

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Create S3 buckets

To import DICOM P10 data into AWS HealthImaging, two Amazon S3 buckets are recommended. The Amazon S3 input bucket stores the DICOM P10 data to be imported and HealthImaging reads from this bucket. The Amazon S3 output bucket stores the processing results of the import job and HealthImaging writes to this bucket. For a visual representation of this, see the diagram at [Understanding import jobs](#).

Note

Due to AWS Identity and Access Management (IAM) policy, your Amazon S3 bucket names must be unique. For more information, see [Bucket naming rules](#) in the *Amazon Simple Storage Service User Guide*.

For the purpose of this guide, we specify the following Amazon S3 input and output buckets in the [IAM role for import](#).

- Input bucket: `arn:aws:s3:::amzn-s3-demo-source-bucket`
- Output bucket: `arn:aws:s3:::amzn-s3-demo-logging-bucket`

For additional information, see [Creating a bucket](#) in the *Amazon S3 User Guide*.

Create a data store

When you import your medical imaging data, the AWS HealthImaging [data store](#) holds the results of your transformed DICOM P10 files, which are called [image sets](#). For a visual representation of this, see the diagram at [Understanding import jobs](#).

Tip

A `datastoreID` is generated when you create a data store. You must use the `datastoreID` when completing the [trust relationship](#) for import later in this section.

To create a data store, see [Creating a data store](#).

Create an IAM user with HealthImaging full access permission

Best practice

We suggest you create separate IAM users for different needs such as importing, data access, and data management. This aligns with [Grant least privilege access](#) in the *AWS Well-Architected Framework*.

For the purposes of the [Tutorial](#) in the next section, you will be using a single IAM user.

To create an IAM user

1. Follow the instructions for [Creating an IAM user in your AWS account](#) in the *IAM User Guide*. Consider naming the user `ahisAdmin` (or similar) for clarification purposes.
2. Assign the `AWSHealthImagingFullAccess` managed policy to the IAM user. For more information, see [AWS managed policy: AWSHealthImagingFullAccess](#).

Note

IAM permissions can be narrowed. For more information, see [AWS managed policies for AWS HealthImaging](#).

Create an IAM role for import

Note

The following instructions refer to an AWS Identity and Access Management (IAM) role that grants read and write access to Amazon S3 buckets for importing your DICOM data. Although the role is required for the [tutorial](#) in the next section, we recommend you add IAM permissions to users, groups, and roles using [AWS managed policies for AWS HealthImaging](#), because they are easier to use than writing policies yourself.

An IAM role is an IAM identity that you can create in your account that has specific permissions. To start an import job, the IAM role that calls the `StartDICOMImportJob` action must be attached to a user policy that grants access to the Amazon S3 buckets used for reading your DICOM P10 data

and storing the import job processing results. It must also be assigned a trust relationship (policy) that enables AWS HealthImaging to assume the role.

To create an IAM role for import purposes

1. Using the [IAM Console](#), create a role named `ImportJobDataAccessRole`. You use this role for the [tutorial](#) in the next section. For more information, see [Creating IAM roles](#) in the *IAM User Guide*.

Tip

For the purposes of this guide, the code examples in [Starting an import job](#) reference the `ImportJobDataAccessRole` IAM role.

2. Attach an IAM permission policy to the IAM role. This permission policy grants access to the Amazon S3 input and output buckets. Attach the following permission policy to the IAM role `ImportJobDataAccessRole`.

```
{
  "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. Attach the following trust relationship (policy) to the `ImportJobDataAccessRole` IAM role. The trust policy requires the `datastoreId` that was generated when you completed the section [Create a data store](#). The [tutorial](#) following this topic assumes you are using one AWS HealthImaging data store, but with data store-specific Amazon S3 buckets, IAM roles, and trust policies.

```

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

```

To learn more about creating and using IAM policies with AWS HealthImaging, see [Identity and Access Management for AWS HealthImaging](#).

To learn more about IAM roles in general, see [IAM roles](#) in the *IAM User Guide*. To learn more about IAM policies and permissions in general, see [IAM Policies and Permissions](#) in the *IAM User Guide*.

Install the AWS CLI (optional)

The following procedure is required if you are using the AWS Command Line Interface. If you're using the AWS Management Console or AWS SDKs, you can skip the following procedure.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*.
 - [Installing or updating the latest version of the AWS CLI](#)
 - [Getting started with the AWS CLI](#)
2. In the AWS CLI config file, add a named profile for the administrator. You use this profile when running the AWS CLI commands. Under the security principle of least privilege, we recommend you create a separate IAM role with privileges specific to the tasks being performed. For more information about named profiles, see [Configuration and credential file settings](#) in the *AWS Command Line Interface User Guide*.

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

3. Verify the setup using the following help command.

```
aws medical-imaging help
```

If the AWS CLI is configured correctly, you see a brief description of AWS HealthImaging and a list of available commands.

AWS HealthImaging tutorial

Objective

The objective of this tutorial is to import DICOM P10 binaries (.dcm files) into an AWS HealthImaging [data store](#) and transform them into [image sets](#) comprised of [metadata](#) and [image](#)

[frames](#) (pixel data). After importing the DICOM data, you use HealthImaging cloud native actions to access the image sets, metadata, and image frames based on your [access preference](#).

Prerequisites

All procedures listed in [Setting up](#) are required to complete this tutorial.

Tutorial steps

1. [Start import job](#)
2. [Get import job properties](#)
3. [Search image sets](#)
4. [Get image set properties](#)
5. [Get image set metadata](#)
6. [Get image set pixel data](#)
7. [Delete data store](#)

Managing data stores with AWS HealthImaging

With AWS HealthImaging, you create and manage [data stores](#) for medical image resources. The following topics describe how to use HealthImaging cloud native actions to create, describe, list, and delete data stores using the AWS Management Console, AWS CLI, and AWS SDKs.

Note

The last topic in this chapter is about [understanding storage tiers](#). After you import your medical imaging data into a HealthImaging data store, it automatically moves between two storage tiers based on time and usage. The storage tiers have different pricing levels, so it's important to understand the tier movement process and the HealthImaging resources that are recognized for billing purposes.

Topics

- [Creating a data store](#)
- [Getting data store properties](#)
- [Listing data stores](#)
- [Deleting a data store](#)
- [Understanding storage tiers](#)

Creating a data store

Use the `CreateDatastore` action to create an AWS HealthImaging [data store](#) for importing DICOM P10 files. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [CreateDatastore](#) in the *AWS HealthImaging API Reference*.

Important

Do not name data stores with protected health information (PHI), personally identifiable information (PII), or other confidential or sensitive information.

To create a data store

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Create data store page](#).
2. Under **Details**, for **Data store name**, enter a name for your data store.
3. Under **Data encryption**, choose an AWS KMS key for encrypting your resources. For more information, see [Data protection in AWS HealthImaging](#).
4. Under **Tags - optional**, you can add tags to your data store when you create it. For more information, see [Tagging a resource](#).
5. Choose **Create data store**.

AWS CLI and SDKs

Bash

AWS CLI with Bash script

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

- For API details, see [CreateDatastore](#) in *AWS CLI Command Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To create a data store

The following `create-datastore` code example creates a data store with the name `my-datastore`.

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

Output:

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

- For API details, see [CreateDatastore](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

    return "";
}
```

- For API details, see [CreateDatastore](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
```

```

export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'CREATING'
  // }
  return response;
};

```

- For API details, see [CreateDatastore](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

```

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

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

```

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [CreateDatastore](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Getting data store properties

Use the `GetDatastore` action to retrieve AWS HealthImaging [data store](#) properties. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [GetDatastore](#) in the *AWS HealthImaging API Reference*.

To get data store properties

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens. Under the **Details** section, all data store properties are available. To view associated image sets, imports, and tags, choose the applicable tab.

AWS CLI and SDKs

Bash

AWS CLI with Bash script

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

- For API details, see [GetDatastore](#) in *AWS CLI Command Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To get a data store's properties

The following `get-datastore` code example gets a data store's properties.

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

Output:

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

- For API details, see [GetDatastore](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

    return null;
}
```

- For API details, see [GetDatastore](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

- For API details, see [GetDatastore](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetDatastore](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Listing data stores

Use the `ListDatastores` action to list available [data stores](#) in AWS HealthImaging. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [ListDatastores](#) in the *AWS HealthImaging API Reference*.

To list data stores

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

- Open the HealthImaging console [Data stores page](#).

All data stores are listed under the **Data stores** section.

AWS CLI and SDKs

Bash

AWS CLI with Bash script

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

- For API details, see [ListDatastores](#) in *AWS CLI Command Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To list data stores

The following `list-datastores` code example lists available data stores.

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

Output:

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

- For API details, see [ListDatastores](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

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

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

```
    }  
  
    return null;  
}
```

- For API details, see [ListDatastores](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";  
import { medicalImagingClient } from "../libs/medicalImagingClient.js";  
  
export const listDatastores = async () => {  
  const paginatorConfig = {  
    client: medicalImagingClient,  
    pageSize: 50,  
  };  
  
  const commandParams = {};  
  const paginator = paginateListDatastores(paginatorConfig, commandParams);  
  
  /**  
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}  
   */  
  const datastoreSummaries = [];  
  for await (const page of paginator) {  
    // Each page contains a list of `jobSummaries`. The list is truncated if is  
    // larger than `pageSize`.  
    datastoreSummaries.push(...page.datastoreSummaries);  
    console.log(page);  
  }  
  // {  
  //   '$metadata': {
```

```
//      statusCode: 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;
};
```

- For API details, see [ListDatastores](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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




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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListDatastores](#) in *AWS SDK for Python (Boto3) API Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Deleting a data store

Use the `DeleteDatastore` action to delete an AWS HealthImaging [data store](#). The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [DeleteDatastore](#) in the *AWS HealthImaging API Reference*.

Note

Before a data store can be deleted, you must first delete all [image sets](#) within it. For more information, see [Deleting an image set](#).

To delete a data store

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.
3. Choose **Delete**.

The **Delete data store** page opens.

4. To confirm data store deletion, enter the data store name in the text input field.
5. Choose **Delete data store**.

AWS CLI and SDKs

Bash

AWS CLI with Bash script

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

- For API details, see [DeleteDatastore](#) in *AWS CLI Command Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To delete a data store

The following delete-datastore code example deletes a data store.

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

Output:

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

- For API details, see [DeleteDatastore](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

- For API details, see [DeleteDatastore](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

  return response;
};
```

- For API details, see [DeleteDatastore](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [DeleteDatastore](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Understanding storage tiers

AWS HealthImaging uses intelligent tiering for automatic clinical lifecycle management. This results in compelling performance and price for both new or active data and long-term archival data with zero friction. HealthImaging bills storage **per GB/month** using the following tiers.

- **Frequent Access Tier** – A tier for frequently accessed data.
- **Archive Instant Access Tier** – A tier for archived data.

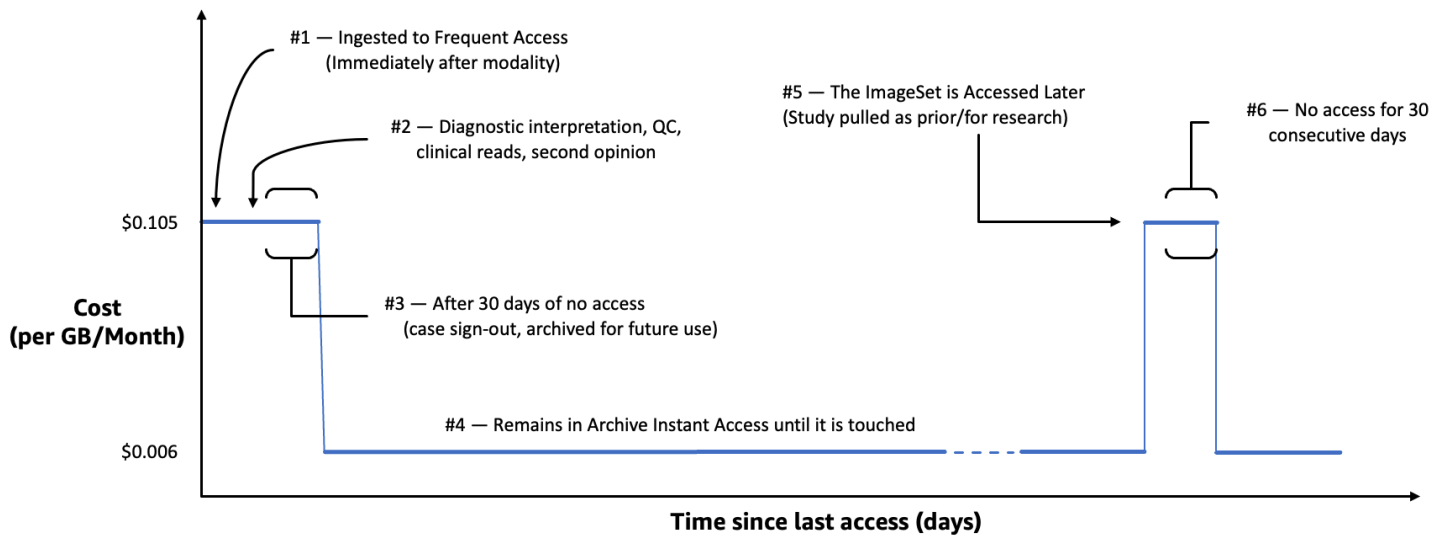
Note

There is not a performance difference between the Frequent Access and Archive Instant Access tiers. Intelligent tiering is applied to specific [image set](#) API actions. Intelligent tiering does not recognize data store, import, and tagging API actions. Movement between the tiers is automatic based on API usage and is explained in the following section.

How does tier movement work?

- After import, image sets start in Frequent Access Tier.
- After **30 consecutive days** of no touches, image sets automatically move to Archive Instant Access Tier.
- Image sets in Archive Instant Access Tier move back to Frequent Access Tier only after being touched.

The following graph provides an overview of the HealthImaging intelligent tiering process.



What is considered a touch?

A touch is a specific API access via the AWS Management Console, AWS CLI, or AWS SDKs and occurs when:

1. A new image set is *created* (`StartDICOMImportJob` or `CopyImageSet`)
2. An image set is *updated* (`UpdateImageSetMetadata` or `CopyImageSet`)
3. An image set's associated metadata or image frame (pixel data) is *read* (`GetImageSetMetaData` or `GetImageFrame`)

The following HealthImaging API actions result in touches and move image sets from Archive Instant Access Tier to Frequent Access Tier.

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

Note

Although [image frames](#) (pixel data) can't be deleted using the `UpdateImageSetMetadata` action, they are still counted for billing purposes.

The following HealthImaging API actions *do not* result in touches. Therefore, they do not move image sets from Archive Instant Access Tier to Frequent Access Tier.

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

Importing imaging data with AWS HealthImaging

Importing is the process of moving your medical imaging data from an Amazon S3 input bucket to an AWS HealthImaging [data store](#). During import, AWS HealthImaging performs a [pixel data verification check](#) before transforming your DICOM P10 files into [image sets](#) comprised of [metadata](#) and [image frames](#) (pixel data).

i Important

HealthImaging import jobs process DICOM instance binaries (.dcm files) and transform them into image sets. Use HealthImaging [cloud native actions](#) (APIs) to manage data stores and image sets. Use HealthImaging's [representation of DICOMweb services](#) to return DICOMweb responses.

The following topics describe how to import your medical imaging data into an HealthImaging data store using the AWS Management Console, AWS CLI, and AWS SDKs.

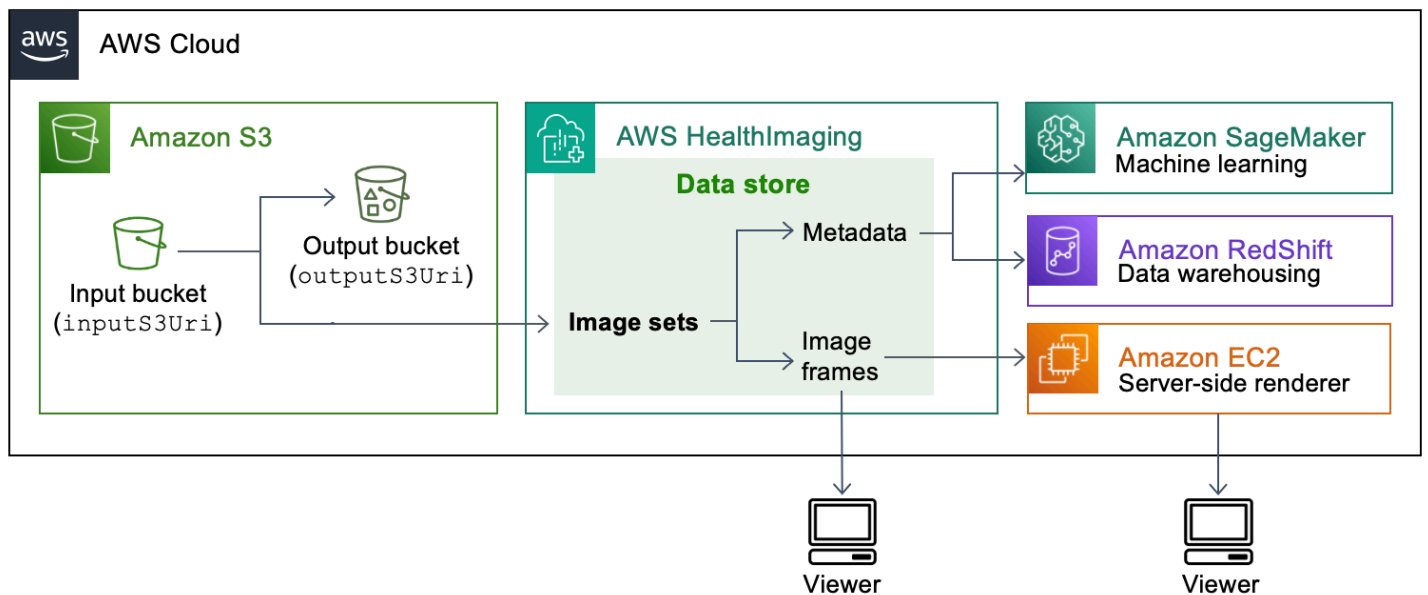
Topics

- [Understanding import jobs](#)
- [Starting an import job](#)
- [Getting import job properties](#)
- [Listing import jobs](#)

Understanding import jobs

After creating a [data store](#) in AWS HealthImaging, you must import your medical imaging data from your Amazon S3 input bucket into your data store to create [image sets](#). You can use the AWS Management Console, AWS CLI, and AWS SDKs to start, describe, and list import jobs.

The following diagram provides an overview of how HealthImaging imports DICOM data into a data store and transforms it into image sets. Import job processing results are stored in the Amazon S3 output bucket (outputS3Uri) and image sets are stored in the AWS HealthImaging data store.



Keep the following points in mind when importing your medical imaging files from Amazon S3 into an AWS HealthImaging data store:

- Specific SOP classes and transfer syntaxes are supported for import jobs. For more information, see [DICOM support](#).
- Length constraints apply to specific DICOM elements during import. To ensure a successful import job, verify that your medical imaging data does not exceed the length constraints. For more information, see [DICOM element constraints](#).
- A pixel data verification check is performed at the beginning of import jobs. For more information, see [Pixel data verification](#).
- There are endpoints, quotas, and throttling limits associated with HealthImaging import actions. For more information, see [Endpoints and quotas](#) and [Throttling limits](#).
- For each import job, processing results are stored at the `outputS3Uri` location. The processing results are organized as a `job-output-manifest.json` file and SUCCESS and FAILURE folders.

Note

You can include up to 10,000 nested folders for a single import job.

- The `job-output-manifest.json` file contains `jobSummary` output and additional details about the processed data. The following example shows output from a `job-output-manifest.json` file.

```
{
  "jobSummary": {
    "jobId": "09876543210987654321098765432109",
    "datastoreId": "12345678901234567890123456789012",
    "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",
    "outputS3Uri": "s3://medical-imaging-output/
job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/",
    "successOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/SUCCESS/",
    "failureOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/FAILURE/",
    "numberOfScannedFiles": 5,
    "numberOfImportedFiles": 3,
    "numberOfFilesWithCustomerError": 2,
    "numberOfFilesWithServerError": 0,
    "numberOfGeneratedImageSets": 2,
    "imageSetsSummary": [{
      "imageSetId": "12345612345612345678907890789012",
      "numberOfMatchedSOPInstances": 2
    },
    {
      "imageSetId": "12345612345612345678917891789012",
      "numberOfMatchedSOPInstances": 1
    }
  ]
}
}
```

- The `SUCCESS` folder holds the `success.ndjson` file containing results of all imaging files that imported successfully. The following example shows output from a `success.ndjson` file.

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

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

- The FAILURE folder holds the `failure.ndjson` file containing results of all imaging files that did not import successfully. The following example shows output from a `failure.ndjson` file.

```
{"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"}}
```

- Import jobs are retained in the list of jobs for 90 days and then archived.

Starting an import job

Use the `StartDICOMImportJob` action to start a [pixel data verification check](#) and bulk data import into an AWS HealthImaging [data store](#). The import job imports DICOM P10 files located in the Amazon S3 input bucket specified by the `inputS3Uri` parameter. The import job processing results are stored in the Amazon S3 output bucket specified by the `outputS3Uri` parameter.

Note

Keep the following points in mind before starting an import job:

- HealthImaging supports importing DICOM P10 files with different transfer syntaxes. Some files retain their original transfer syntax encoding during import, while others are transcoded to HTJ2K lossless by default. For more information, see [Supported transfer syntaxes](#).
- HealthImaging supports data imports from Amazon S3 buckets located in other [supported Regions](#). To achieve this functionality, provide the `inputOwnerAccountId` parameter when starting an import job. For more information, see [Cross-account import for AWS HealthImaging](#).
- HealthImaging applies length constraints to specific DICOM elements during import. For more information, see [DICOM element constraints](#).

The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [StartDICOMImportJob](#) in the *AWS HealthImaging API Reference*.

To start an import job

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.
3. Choose **Import DICOM data**.

The **Import DICOM data** page opens.

4. Under the **Details** section, enter the following information:
 - **Name** (optional)
 - **Import source location in S3**
 - **Source bucket owner's account ID** (optional)
 - **Encryption key** (optional)
 - **Output destination in S3**
5. Under the **Service access** section, choose **Use an existing service role** and select the role from the **Service role name** menu or choose **Create and use a new service role**.
6. Choose **Import**.

AWS CLI and SDKs

C++

SDK for C++

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

```

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

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

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

```



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

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To start a dicom import job

The following `start-dicom-import-job` code example starts a dicom import job.

```
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"
```

Output:

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

- For API details, see [StartDICOMImportJob](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

    try {
        StartDicomImportJobRequest startDicomImportJobRequest =
StartDicomImportJobRequest.builder()
            .jobName(jobName)
            .datastoreId(datastoreId)
            .dataAccessRoleArn(dataAccessRoleArn)
            .inputS3Uri(inputS3Uri)
            .outputS3Uri(outputS3Uri)
            .build();

        StartDicomImportJobResponse response =
medicalImagingClient.startDICOMImportJob(startDicomImportJobRequest);
        return response.jobId();
    } catch (MedicalImagingException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return "";
}
```

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

```

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

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Getting import job properties

Use the `GetDICOMImportJob` action to learn more about AWS HealthImaging import job properties. For instance, after starting an import job, you can run `GetDICOMImportJob` to find the status of the job. Once the `jobStatus` returns as `COMPLETED`, you're ready to access your [image sets](#).

Note

The `jobStatus` refers to the execution of the import job. Therefore, an import job can return a `jobStatus` as `COMPLETED` even if validation issues are discovered during the import process. If a `jobStatus` returns as `COMPLETED`, we still recommend you review the output manifests written to Amazon S3, as they provide details on the success or failure of individual P10 object imports.

The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [GetDICOMImportJob](#) in the *AWS HealthImaging API Reference*.

To get import job properties

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens. The **Image sets** tab is selected by default.

3. Choose the **Imports** tab.
4. Choose an import job.

The **Import job details** page opens and displays properties about the import job.

AWS CLI and SDKs

C++

SDK for C++

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

    return outcome;
}
```

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To get a dicom import job's properties

The following `get-dicom-import-job` code example gets a dicom import job's properties.

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

Output:

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

- For API details, see [GetDICOMImportJob](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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



```
    try {
        GetDicomImportJobRequest getDicomImportJobRequest =
        GetDicomImportJobRequest.builder()
            .datastoreId(datastoreId)
            .jobId(jobId)
            .build();

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

    return null;
}
```

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

```
const response = await medicalImagingClient.send(
  new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
);
console.log(response);
// {
//   '$metadata': {
//     statusCode: 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     endedAt: 2023-09-19T17:29:21.753Z,
//     inputS3Uri: 's3://healthimaging-source/CTStudy/',
//     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     jobName: 'job_1',
//     jobStatus: 'COMPLETED',
//     outputS3Uri: 's3://health-imaging-dest/
output_ct/'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'/',
//     submittedAt: 2023-09-19T17:27:25.143Z
//   }
// }

return response;
};
```

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Listing import jobs

Use the `ListDICOMImportJobs` action to list import jobs created for a specific HealthImaging [data store](#). The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [ListDICOMImportJobs](#) in the *AWS HealthImaging API Reference*.

Note

Import jobs are retained in the list of jobs for 90 days and then archived.

To list import jobs

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens. The **Image sets** tab is selected by default.

3. Choose the **Imports** tab to list all associated import jobs.

AWS CLI and SDKs

CLI

AWS CLI

To list dicom import jobs

The following `list-dicom-import-jobs` code example lists dicom import jobs.

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

Output:

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

- For API details, see [ListDICOMImportJobs](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

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

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

- For API details, see [ListDICOMImportJobs](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
    datastoreId = "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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       endedAt: 2023-09-22T14:49:51.351Z,
//       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       jobName: 'test-1',
//       jobStatus: 'COMPLETED',
//       submittedAt: 2023-09-22T14:48:45.767Z
//     }
//   ]
// }

return jobSummaries;
};

```

- For API details, see [ListDICOMImportJobs](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListDICOMImportJobs](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Accessing image sets with AWS HealthImaging

Accessing medical imaging data in AWS HealthImaging typically involves searching for an [image set](#) with a unique key and getting the associated [metadata](#) and [image frames](#) (pixel data).

Important

During import, HealthImaging processes DICOM instance binaries (.dcm files) and transforms them into image sets. Use HealthImaging [cloud native actions](#) (APIs) to manage data stores and image sets. Use HealthImaging's [representation of DICOMweb services](#) to return DICOMweb responses.

The following topics explain how to use HealthImaging cloud native actions in the AWS Management Console, AWS CLI, and AWS SDKs to search image sets and get their associated properties, metadata, and image frames.

Topics

- [Understanding image sets](#)
- [Searching image sets](#)
- [Getting image set properties](#)
- [Getting image set metadata](#)
- [Getting image set pixel data](#)

Understanding image sets

Image sets are an AWS concept that serve as the foundation for AWS HealthImaging. Image sets are created when you import your DICOM data into HealthImaging, so having a good understanding of them is required when working with the service.

Image sets were introduced for the following reasons:

- Support a wide variety of medical imaging workflows (clinical and nonclinical) through flexible APIs.
- Maximize patient safety by grouping only related data.

- Encourage data to be cleaned to help increase the visibility of inconsistencies. For more information, see [Modifying image sets](#).

Important

Clinical use of DICOM data before it has been cleaned can result in patient harm.

The following menus describe image sets in further detail and provide examples and diagrams to help you comprehend their functionality and purpose in HealthImaging.

What is an image set?

An image set is an AWS concept that defines an abstract grouping mechanism for optimizing related medical imaging data. When you import your DICOM P10 imaging data into an AWS HealthImaging data store, it is transformed into image sets comprised of [metadata](#) and [image frames](#) (pixel data).

Note

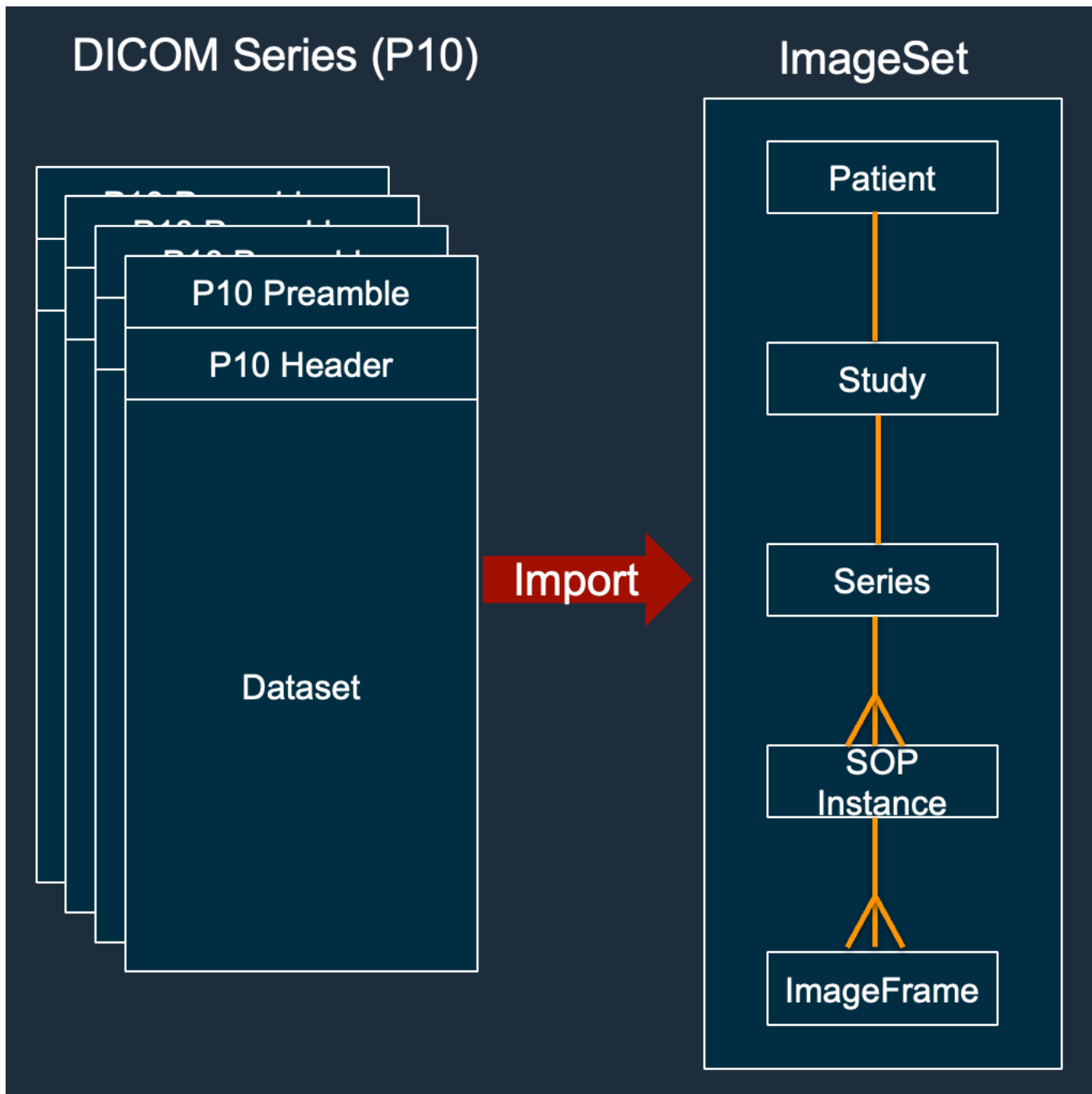
Image set metadata is [normalized](#). In other words, one common set of attributes and values maps to Patient, Study, and Series level elements listed in the [Registry of DICOM Data Elements](#).

During import, some image sets retain their original transfer syntax encoding, while others are transcoded to High-Throughput JPEG 2000 (HTJ2K) lossless by default. If an image set is encoded in HTJ2K, it must be decoded prior to viewing. For more information, see [Supported transfer syntaxes](#) and [HTJ2K decoding libraries](#).

Image frames (pixel data) are encoded in High-Throughput JPEG 2000 (HTJ2K) and must be [decoded](#) prior to viewing.

Image sets are AWS resources, so they are assigned [Amazon Resource Names \(ARNs\)](#). They can be tagged with up to 50 key-value pairs and granted [role-based access control \(RBAC\)](#) and [attribute-based access control \(ABAC\)](#) through IAM. In addition, image sets are [versioned](#), so all changes are preserved and prior versions can be accessed.

Importing DICOM P10 data results in image sets that contain DICOM metadata and image frames for one or more Service-Object Pair (SOP) instances in the same DICOM Series.

**Note**

DICOM import jobs:

- *Always* create new image sets and *never* update existing image sets.
- Do not deduplicate SOP Instance storage, as each import of the same SOP Instance uses additional storage.

- *May* create multiple image sets for a single DICOM Series. For example, when there is a variant of a [normalized metadata attribute](#) such as a PatientName mismatch.

What does image set metadata look like?

Use the `GetImageSetMetadata` action to retrieve image set metadata. The returned metadata is compressed with `gzip`, so you must unzip it before viewing. For more information, see [Getting image set metadata](#).

The following example shows the structure of image set [metadata](#) in JSON format.

```
{
  "SchemaVersion": "1.1",
  "DatastoreID": "2aa75d103f7f45ab977b0e93f00e6fe9",
  "ImageSetID": "46923b66d5522e4241615ecd64637584",
  "Patient": {
    "DICOM": {
      "PatientBirthDate": null,
      "PatientSex": null,
      "PatientID": "2178309",
      "PatientName": "MISTER^CT"
    }
  },
  "Study": {
    "DICOM": {
      "StudyTime": "083501",
      "PatientWeight": null
    }
  },
  "Series": {
    "1.2.840.113619.2.30.1.1762295590.1623.978668949.887": {
      "DICOM": {
        "Modality": "CT",
        "PatientPosition": "FFS"
      }
    },
    "Instances": {
      "1.2.840.113619.2.30.1.1762295590.1623.978668949.888": {
        "DICOM": {
          "SourceApplicationEntityTitle": null,
          "SOPClassUID": "1.2.840.10008.5.1.4.1.1.2",
          "HighBit": 15,
          "PixelData": null,

```

```
"Exposure": "40",
"RescaleSlope": "1",
"ImageFrames": [
  {
    "ID": "0d1c97c51b773198a3df44383a5fd306",
    "PixelDataChecksumFromBaseToFullResolution": [
      {
        "Width": 256,
        "Height": 188,
        "Checksum": 2598394845
      },
      {
        "Width": 512,
        "Height": 375,
        "Checksum": 1227709180
      }
    ],
    "MinPixelValue": 451,
    "MaxPixelValue": 1466,
    "FrameSizeInBytes": 384000
  }
]
}
```

Image set creation example: multiple import jobs

The following example shows how multiple import jobs always create new image sets and *never* add to existing ones.

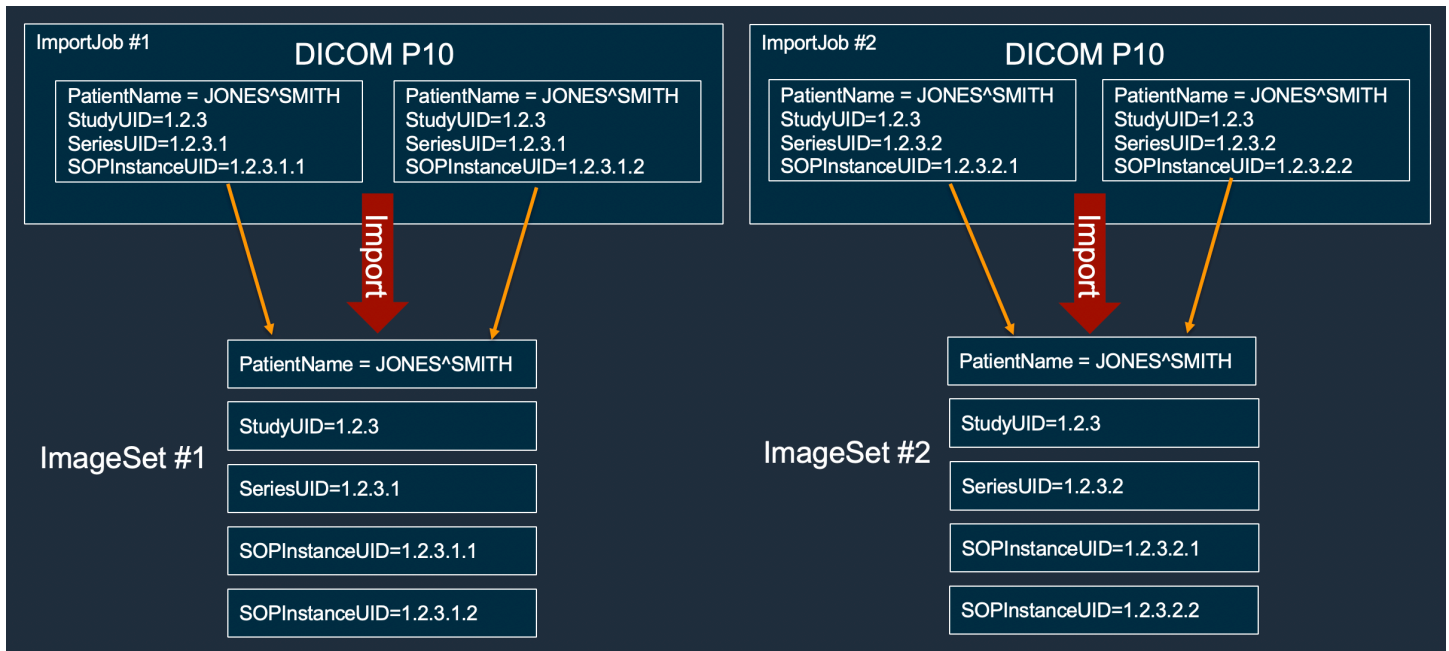


Image set creation example: single import job with two variants

The following example shows a single import job creating two image sets because instances 1 and 2 have different patient names than instances 3 and 4.

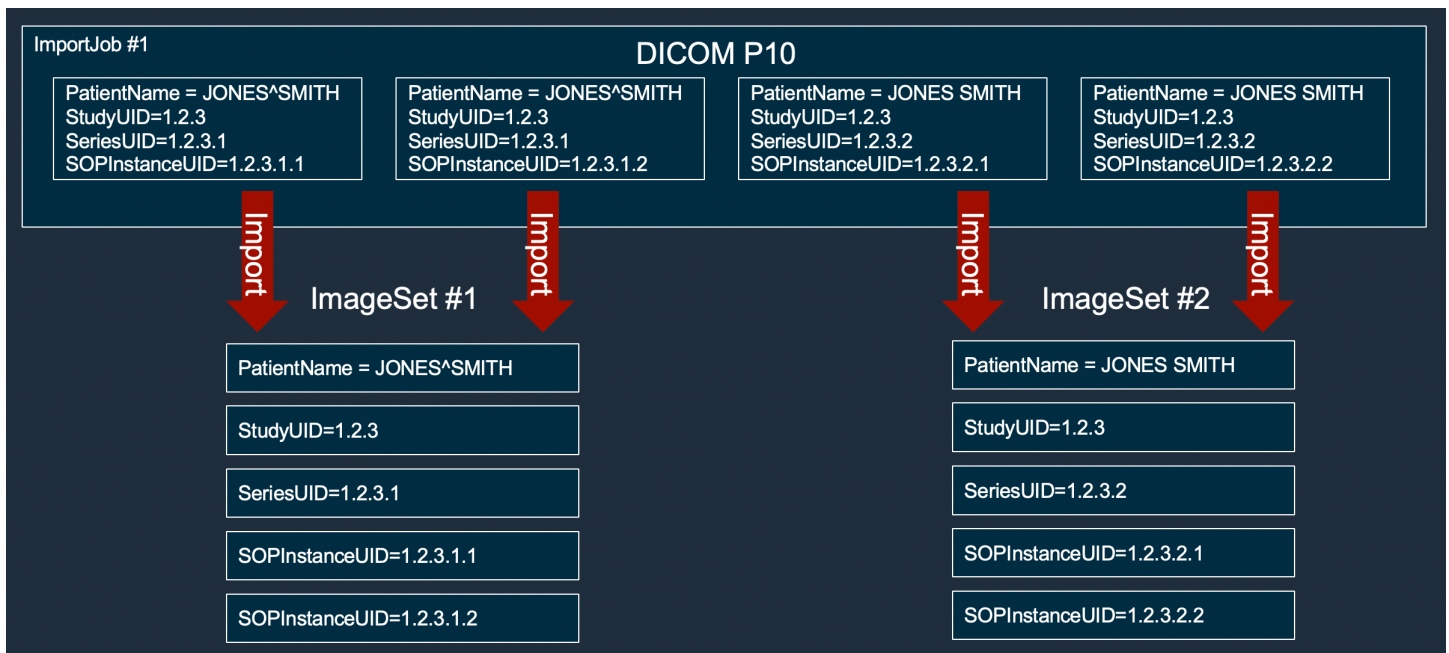
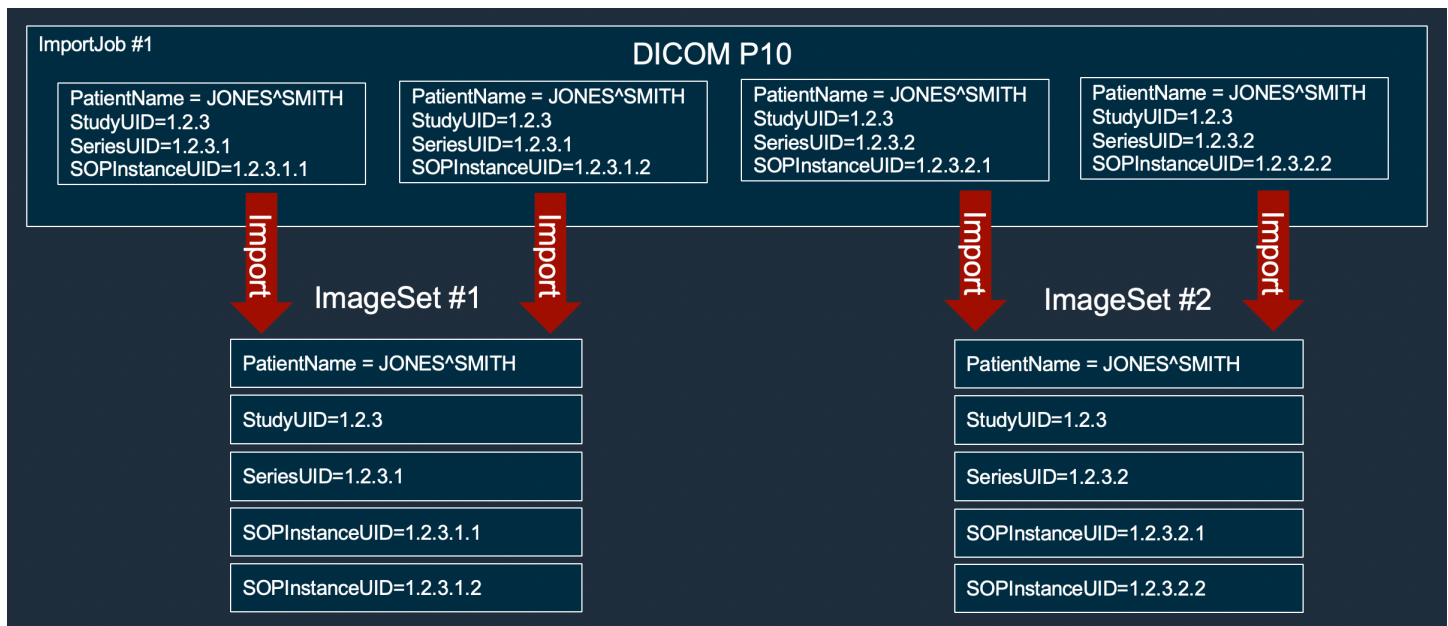


Image set creation example: single import job with optimization

The following example shows a single import job creating two image sets to improve throughput, even though the patient names match.



Searching image sets

Use the `SearchImageSets` action to run search queries against all [image sets](#) in an ACTIVE HealthImaging data store. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [SearchImageSets](#) in the *AWS HealthImaging API Reference*.

Note

Keep the following points in mind when searching image sets.

- `SearchImageSets` accepts a single search query parameter and returns a paginated response of all image sets that have the matching criteria. All date range queries must be input as (`lowerBound`, `upperBound`).
- By default, `SearchImageSets` uses the `updatedAt` field for sorting in decreasing order from newest to oldest.
- If you created your data store with a customer-owned AWS KMS key, you must update your AWS KMS key policy before interacting with image sets. For more information, see [Creating a customer managed key](#).

To search image sets

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

Note

The following procedures show how to search image sets using the `Series Instance UID` and `Updated at` property filters.

Series Instance UID

Search image sets using the `Series Instance UID` property filter

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens and the **Image sets** tab is selected by default.

3. Choose the property filter menu and select `Series Instance UID`.
4. In the **Enter value to search** field, enter (paste) the Series Instance UID of interest.

Note

Series Instance UID values must be identical to those listed in the [Registry of DICOM Unique Identifiers \(UIDs\)](#). Note the requirements include a series of numbers that contain at least one period between them. Periods are not allowed at the beginning or end of Series Instance UIDs. Letters and white space are not allowed, so use caution when copying and pasting UIDs.

5. Choose the **Date range** menu, select a date range for the Series Instance UID, and choose **Apply**.
6. Choose **Search**.

Series Instance UIDs that fall within the selected date range are returned in Newest order by default.

Updated at

Search image sets using the Updated at property filter

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens and the **Image sets** tab is selected by default.

3. Choose the property filter menu and choose Updated at.
4. Choose the **Date range** menu, select an image set date range, and choose **Apply**.
5. Choose **Search**.

Image sets that fall within the selected date range are returned in Newest order by default.

AWS CLI and SDKs

C++

SDK for C++

The utility function for searching image sets.

```
//! 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;
}

```

Use case #1: EQUAL operator.

```

    Aws::Vector<Aws::String> imageIDsForPatientID;
    Aws::MedicalImaging::Model::SearchCriteria searchCriteriaEqualsPatientID;
    Aws::Vector<Aws::MedicalImaging::Model::SearchFilter>
patientIDSearchFilters = {

    Aws::MedicalImaging::Model::SearchFilter().WithOperator(Aws::MedicalImaging::Model::Oper

    .WithValues({Aws::MedicalImaging::Model::SearchByAttributeValue().WithDICOMPatientId(pat
        });
}

```

```

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

searchCriteriaEqualsPatientID,

imageIDsForPatientID,

                                                                    clientConfig);

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

```

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```

        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;
        }
    }
}

```

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```

    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 : useCase3Results) {
            std::cout << " Image set with ID '" << imageSetResult <<
std::endl;
        }
    }
}

```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```

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

    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;
        }
    }
}

```

- For API details, see [SearchImageSets](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

Example 1: To search image sets with an EQUAL operator

The following `search-image-sets` code example uses the EQUAL operator to search image sets based on a specific value.

```

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

```

Contents of `search-criteria.json`

```
{
```

```

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

```

Output:

```

{
  "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"
  }]
}

```

Example 2: To search image sets with a BETWEEN operator using DICOMStudyDate and DICOMStudyTime

The following `search-image-sets` code example searches for image sets with DICOM Studies generated between January 1, 1990 (12:00 AM) and January 1, 2023 (12:00 AM).

Note: `DICOMStudyTime` is optional. If it is not present, 12:00 AM (start of the day) is the time value for the dates provided for filtering.

```

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

```


Contents of search-criteria.json

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

Output:

```
{
  "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"
  ]
}
```

Example 3: To search image sets with a BETWEEN operator using createdAt (time studies were previously persisted)

The following search-image-sets code example searches for image sets with DICOM Studies persisted in HealthImaging between the time ranges in UTC time zone.

Note: Provide createdAt in example format ("1985-04-12T23:20:50.52Z").

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

Contents of search-criteria.json

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

Output:

```
{  
  "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"
}]
}

```

Example 4: To search image sets with an EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field

The following search-image-sets code example searches for image sets with an EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

Note: Provide updatedAt in example format ("1985-04-12T23:20:50.52Z").

```

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

```

Contents of 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"
  }
}

```

Output:

```
{
  "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"
  ]
}
```

- For API details, see [SearchImageSets](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

The utility function for searching image sets.

```
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;
}

```

Use case #1: EQUAL operator.

```

        List<SearchFilter> searchFilters =
Collections.singletonList(SearchFilter.builder()
            .operator(Operator.EQUAL)
            .values(SearchByAttributeValue.builder()
                .dicomPatientId(patientId)
                .build())
            .build());

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

        List<ImageSetsMetadataSummary> imageSetsMetadataSummaries =
searchMedicalImagingImageSets(
            medicalImagingClient,
            datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println("The image sets for patient " + patientId + " are:
\n"
                + imageSetsMetadataSummaries);
            System.out.println();
        }
    }
}

```

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```

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

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

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

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

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

```

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```

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();
    }

```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```

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();
        }
    }
}

```

- For API details, see [SearchImageSets](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

The utility function for searching image sets.

```

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;
};
```

Use case #1: EQUAL operator.

```
const datastoreId = "12345678901234567890123456789012";

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

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

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```
const datastoreId = "12345678901234567890123456789012";

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

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

```

        DICOMStudyTime: "000000",
      },
    ],
    operator: "BETWEEN",
  },
],
};

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

```

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```

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);
}

```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```

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);
}
```

- For API details, see [SearchImageSets](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

The utility function for searching image sets.

```
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
```

Use case #1: EQUAL operator.

```

search_filter = {
    "filters": [
        {"operator": "EQUAL", "values": [{"DICOMPatientId": patient_id}]}
    ]
}

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

```

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```

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

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

```

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```

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}"
)

```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```

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}")
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [SearchImageSets](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Getting image set properties

Use the `GetImageSet` action to return properties for a given [image set](#) in HealthImaging. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [GetImageSet](#) in the *AWS HealthImaging API Reference*.

Note

By default, AWS HealthImaging returns properties for the latest version of an image set. To view properties for an older version of an image set, provide the `versionId` with your request.

Use `GetDICOMInstance`, HealthImaging's representation of a DICOMweb service, to return a DICOM instance binary (.dcm file). For more information, see [Getting a DICOM instance from HealthImaging](#).

To get image set properties

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens and the **Image sets** tab is selected by default.

3. Choose an image set.

The **Image set details** page opens and displays image set properties.

AWS CLI and SDKs

CLI

AWS CLI

To get image set properties

The following `get-image-set` code example gets the properties for an image set.

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

Output:

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

- For API details, see [GetImageSet](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

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

    return null;
}
```

- For API details, see [GetImageSet](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
    datastoreId = "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: 'xxxxxxxxxxxxxxxx',
  //   imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/imageSet/xxxxxxxxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxx',
  //   imageSetState: 'ACTIVE',
  //   imageSetWorkflowStatus: 'CREATED',
  //   updatedAt: 2023-09-22T14:49:26.427Z,
  //   versionId: '1'
  // }

  return response;
};
```

- For API details, see [GetImageSet](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

```
self.health_imaging_client = health_imaging_client

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

    :param datastore_id: The ID of the data store.
    :param image_set_id: The ID of the image set.
    :param version_id: The optional version of the image set.
    :return: The image set properties.
    """
    try:
        if version_id:
            image_set = self.health_imaging_client.get_image_set(
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                versionId=version_id,
            )
        else:
            image_set = self.health_imaging_client.get_image_set(
                imageSetId=image_set_id, datastoreId=datastore_id
            )
    except ClientError as err:
        logger.error(
            "Couldn't get image set. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return image_set
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetImageSet](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Getting image set metadata

Use the `GetImageSetMetadata` action to retrieve [metadata](#) for a given [image set](#) in HealthImaging. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [GetImageSetMetadata](#) in the *AWS HealthImaging API Reference*.

Note

By default, HealthImaging returns metadata attributes for the latest version of an image set. To view metadata for an older version of an image set, provide the `versionId` with your request.

Image set metadata is compressed with `gzip` and returned as a JSON object. Therefore, you must decompress the JSON object prior to viewing the normalized metadata. For more information, see [Metadata normalization](#).

Use `GetDICOMInstanceMetadata`, HealthImaging's representation of a DICOMweb service, to return DICOM instance metadata (`.json` file). For more information, see [Getting DICOM instance metadata from HealthImaging](#).

To get image set metadata

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens and the **Image sets** tab is selected by default.

3. Choose an image set.

The **Image set details** page opens and the image set metadata displays under the **Image set metadata viewer** section.

AWS CLI and SDKs

C++

SDK for C++

Utility function to get image set metadata.

```
#!/ Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The HealthImaging image set ID.
  \param versionID: The HealthImaging image set version ID, ignored if empty.
  \param outputPath: The path where the metadata will be stored as gzipped
  json.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
&outputFilePath,
                                                  const
  Aws::Client::ClientConfiguration &clientConfig) {
  Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
  request.SetDatastoreId(dataStoreID);
  request.SetImageSetId(imageSetID);
  if (!versionID.empty()) {
    request.SetVersionId(versionID);
  }
}
```

```

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

    return outcome.IsSuccess();
}

```

Get image set metadata without version.

```

    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;
    }

```

Get image set metadata with version.

```

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

```

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI**Example 1: To get image set metadata without version**

The following `get-image-set-metadata` code example gets metadata for an image set without specifying a version.

Note: `outfile` is a required parameter

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

The returned metadata is compressed with gzip and stored in the `studymetadata.json.gz` file. To view the contents of the returned JSON object, you must first decompress it.

Output:

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

Example 2: To get image set metadata with version

The following `get-image-set-metadata` code example gets metadata for an image set with a specified version.

Note: `outfile` is a required parameter

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

```
--datastore-id 12345678901234567890123456789012 \  
--image-set-id ea92b0d8838c72a3f25d00d13616f87e \  
--version-id 1 \  
studymetadata.json.gz
```

The returned metadata is compressed with gzip and stored in the `studymetadata.json.gz` file. To view the contents of the returned JSON object, you must first decompress it.

Output:

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

- For API details, see [GetImageSetMetadata](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static void getMedicalImageSetMetadata(MedicalImagingClient  
medicalImagingClient,  
    String destinationPath,  
    String datastoreId,  
    String imagesetId,  
    String versionId) {  
  
    try {  
        GetImageSetMetadataRequest.Builder getImageSetMetadataRequestBuilder  
= GetImageSetMetadataRequest.builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imagesetId);  
  
        if (versionId != null) {  
            getImageSetMetadataRequestBuilder =  
getImageSetMetadataRequestBuilder.versionId(versionId);  
        }  
  
        medicalImagingClient.getImageSetMetadata(getImageSetMetadataRequestBuilder.build()),
```

```

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

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

```

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

Utility function to get image set metadata.

```

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;
};
```

Get image set metadata without version.

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

Get image set metadata with version.

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

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

Utility function to get image set metadata.

```
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
```

Get image set metadata without version.

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

Get image set metadata with version.

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Getting image set pixel data

An [image frame](#) is the pixel data that exists within an image set to make up a 2D medical image. Use the `GetImageFrame` action to retrieve an HTJ2K-encoded image frame for a given [image set](#) in HealthImaging. The following menus provide code examples for the AWS CLI and AWS SDKs. For more information, see [GetImageFrame](#) in the *AWS HealthImaging API Reference*.

Note

Keep the following points in mind when using the `GetImageFrame` action:

- During [import](#), HealthImaging retains encoding for some transfer syntaxes, but transcodes others to HTJ2K lossless by default. Therefore, image frames must be decoded prior to viewing in an image viewer. For more information, see [Supported transfer syntaxes](#) and [HTJ2K decoding libraries](#).
- The `GetImageFrame` action returns the image frame in the stored transfer syntax of the instance by default. For more information, see [Supported transfer syntaxes](#).
- You can also use `GetDICOMInstanceFrames`, HealthImaging's representation of a DICOMweb service, to retrieve DICOM instance frames (multipart request) for DICOMweb-compatible viewers and applications. For more information, see [Getting DICOM instance frames from HealthImaging](#).

To get image set pixel data

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

Note

Image frames must be accessed and decoded programmatically, as an image viewer is not available in the AWS Management Console.

For more information about decoding and viewing image frames, see [HTJ2K decoding libraries](#).

AWS CLI and SDKs

C++

SDK for C++

```
#!/ Routine which downloads an AWS HealthImaging image frame.
/*!
  \param datastoreID: The HealthImaging data store ID.
  \param imageSetID: The image set ID.
  \param frameID: The image frame ID.
  \param jphFile: File to store the downloaded frame.
```



```
\param clientConfig: Aws client configuration.
\return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageFrame(const Aws::String &dataStoreID,
                                             const Aws::String &imageSetID,
                                             const Aws::String &frameID,
                                             const Aws::String &jphFile,
                                             const
Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);

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

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

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

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

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

    return outcome.IsSuccess();
}
```

- For API details, see [GetImageFrame](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI**To get image set pixel data**

The following `get-image-frame` code example gets an image frame.

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

Note: This code example does not include output because the `GetImageFrame` action returns a stream of pixel data to the `imageframe.jpg` file. For information about decoding and viewing image frames, see HTJ2K decoding libraries.

- For API details, see [GetImageFrame](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static void getMedicalImageSetFrame(MedicalImagingClient  
medicalImagingClient,  
    String destinationPath,  
    String datastoreId,  
    String imagesetId,  
    String imageFrameId) {  
  
    try {  
        GetImageFrameRequest getImageSetMetadataRequest =  
        GetImageFrameRequest.builder()
```

```

                .datastoreId(datastoreId)
                .imageSetId(imagesetId)

        .imageFrameInformation(ImageFrameInformation.builder()

        .imageFrameId(imageFrameId)

                .build())

        .build();

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

```

- For API details, see [GetImageFrame](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

```

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

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

```

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

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

- For API details, see [GetImageFrame](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetImageFrame](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Modifying image sets with AWS HealthImaging

DICOM import jobs typically require you to modify your [image sets](#) for the following reasons:

- Patient safety
- Data consistency
- Reduce storage costs

Important

During import, HealthImaging processes DICOM instance binaries (.dcm files) and transforms them into image sets. Use HealthImaging [cloud native actions](#) (APIs) to manage data stores and image sets. Use HealthImaging's [representation of DICOMweb services](#) to return DICOMweb responses.

HealthImaging provides several cloud native APIs to simplify the image set modification process. The following topics describe how to modify image sets using AWS CLI and AWS SDKs.

Topics

- [Listing image set versions](#)
- [Updating image set metadata](#)
- [Copying an image set](#)
- [Deleting an image set](#)

Listing image set versions

Use the `ListImageSetVersions` action to list version history for an [image set](#) in HealthImaging. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [ListImageSetVersions](#) in the *AWS HealthImaging API Reference*.

Note

AWS HealthImaging records every change made to an image set. Updating image set [metadata](#) creates a new version in the image set history. For more information, see [Updating image set metadata](#).

To list versions for an image set

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens and the **Image sets** tab is selected by default.

3. Choose an image set.

The **Image set details** page opens.

The image set **Version** displays under the **Image set details** section.

AWS CLI and SDKs

CLI

AWS CLI**To list image set versions**

The following `list-image-set-versions` code example lists the version history for an image set.

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

Output:


```
{
  "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
    }
  ]
}
```

- For API details, see [ListImageSetVersions](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

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

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

    return null;
}
```

- For API details, see [ListImageSetVersions](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      imageSetState: 'ACTIVE',
//      versionId: '1'
//    }]
// }
return imageSetPropertiesList;
};

```

- For API details, see [ListImageSetVersions](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

```

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

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

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

```

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListImageSetVersions](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Updating image set metadata

Use the `UpdateImageSetMetadata` action to update image set [metadata](#) in AWS HealthImaging. You can use this asynchronous process to add, update, and remove image set metadata attributes,

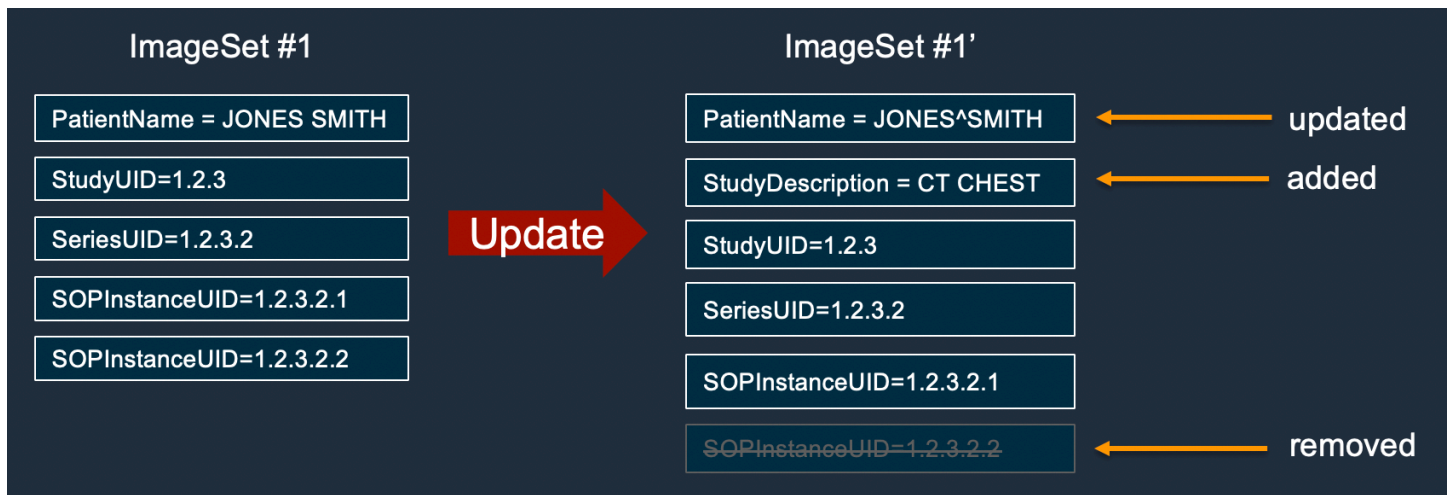
which are manifestations of [DICOM normalization elements](#) that are created during import. Using the `UpdateImageSetMetadata` action, you can also remove Series and SOP Instances to keep image sets in sync with external systems and to de-identify image set metadata. For more information, see [UpdateImageSetMetadata](#) in the *AWS HealthImaging API Reference*.

Note

Real-world DICOM imports require updating, adding, and removing attributes from the image set metadata. Keep the following points in mind when updating image set metadata:

- Updating image set metadata creates a new version in the image set history. For more information, see [Listing image set versions](#). To revert to a previous image set version ID, use the optional [revertToVersionId](#) parameter.
- Updating image set metadata is an asynchronous process. Therefore, [imageSetState](#) and [imageSetWorkflowStatus](#) response elements are available to provide the respective state and status of an image set undergoing update. You cannot perform other write operations on a LOCKED image set.
- If the `UpdateImageSetMetadata` action is not successful, call and review the [message](#) response element to see [common errors](#).
- DICOM element constraints are applied to metadata updates. The [force](#) request parameter allows you to update elements in cases where you want to override [DICOM metadata constraints](#).
- Set the [force](#) request parameter to force completion of the `UpdateImageSetMetadata` action. Setting this parameter allows the following updates to an image set:
 - Updating the `Tag.StudyInstanceUID`, `Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID`, and `Tag.StudyID` attributes
 - Adding, removing, or updating instance level private DICOM data elements

The following diagram represents image set metadata being updated in HealthImaging.



To update image set metadata

Choose a tab based on your access preference to AWS HealthImaging.

AWS CLI and SDKs

CLI

AWS CLI

Example 1: To insert or update an attribute in image set metadata

The following `update-image-set-metadata` example inserts or updates an attribute in image set metadata.

```
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
```

Contents of `metadata-updates.json`

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

```
}

```

Output:

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

Example 2: To remove an attribute from image set metadata

The following `update-image-set-metadata` example removes an attribute from image set metadata.

```
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
```

Contents of `metadata-updates.json`

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

Output:

```
{
  "latestVersionId": "2",
  "imageSetWorkflowStatus": "UPDATING",
  "updatedAt": 1680042257.908,
```



```

    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
  }

```

Example 3: To remove an instance from image set metadata

The following `update-image-set-metadata` example removes an instance from image set metadata.

```

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

```

Contents of `metadata-updates.json`

```

{
  "DICOMUpdates": {
    "removableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}}"
  }
}

```

Output:

```

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

```

Example 4: To revert an image set to a previous version

The following `update-image-set-metadata` example shows how to revert an image set to a prior version. `CopyImageSet` and `UpdateImageSetMetadata` actions create new versions of image sets.

```
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"}'
```

Output:

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

Example 5: To add a private DICOM data element to an instance

The following `update-image-set-metadata` example shows how to add a private element to a specified instance within an image set. The DICOM standard permits private data elements for communication of information that cannot be contained in standard data elements. You can create, update, and delete private data elements with the `UpdateImageSetMetadata` action.

```
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
```

Contents of `metadata-updates.json`

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series
\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances
\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":
{\"001910F9\": \"97\"}, \"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}}"
  }
}
```

Output:

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

Example 6: To update a private DICOM data element to an instance

The following `update-image-set-metadata` example shows how to update the value of a private data element belonging to an instance within an image set.

```
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
```

Contents of `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\"}}}}}}}"
  }
}
```

```
}
}
```

Output:

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

Example 7: To update a SOPInstanceUID with the force parameter

The following `update-image-set-metadata` example shows how to update a SOPInstanceUID, using the `force` parameter to override the DICOM metadata constraints.

```
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
```

Contents of `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\\"}}}}}}}"
  }
}
```

Output:

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

- For API details, see [UpdateImageSetMetadata](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

```

        .datastoreId(datastoreId)
        .imageSetId(imageSetId)
        .latestVersionId(versionId)
        .updateImageSetMetadataUpdates(metadataUpdates)
        .force(force)
        .build();

    UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

```

Use case #1: Insert or update an attribute.

```

    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);

```

Use case #2: Remove an attribute.

```

        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);

```

Use case #3: Remove an instance.

```

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

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                    "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                }
            }
        }
    }
}

```

```

        """;
        MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
            .dicomUpdates(DICOMUpdates.builder()
                .removableAttributes(SdkBytes.fromByteBuffer(
                    ByteBuffer.wrap(removeInstance
.getBytes(StandardCharsets.UTF_8))))
            .build())
        .build();

        updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
            imagesetId,
                versionid, metadataRemoveUpdates, force);

```

Use case #4: Revert to a previous version.

```

        // 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);

```

- For API details, see [UpdateImageSetMetadata](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

```

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

```



```
/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 * version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 200,
    //     requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //     extendedRequestId: undefined,
    //     cfId: undefined,
    //     attempts: 1,
    //     totalRetryDelay: 0
    //   },
    //   createdAt: 2023-09-22T14:49:26.427Z,
    //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   imageSetState: 'LOCKED',
    //   imageSetWorkflowStatus: 'UPDATING',
    //   latestVersionId: '4',
    //   updatedAt: 2023-09-27T19:41:43.494Z
    // }
```

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

Use case #1: Insert or update an attribute and force the update.

```
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,
);
```

Use case #2: Remove an attribute.

```
// 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,
);
```

Use case #3: Remove an instance.

```
const remove_instance = JSON.stringify({
  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": {},
        },
      },
    },
  },
});

const updateMetadata = {
  DICOMUpdates: {
    removableAttributes: new TextEncoder().encode(remove_instance),
  },
};

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

Use case #4: Revert to an earlier version.

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

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

- For API details, see [UpdateImageSetMetadata](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
```

```

        :param metadata: The image set metadata as a dictionary.
            For example {"DICOMUpdates": {"updateableAttributes":
                {"\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

```

The following code instantiates the `MedicalImagingWrapper` object.

```

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

```

Use case #1: Insert or update an attribute.

```

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
    )

```

Use case #2: Remove an attribute.

```

# 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
)

```

Use case #3: Remove an instance.

```

attributes = ""{
    "SchemaVersion": 1.1,
    "Study": {
        "Series": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
            }
        }
    }
}""
metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

```

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

Use case #4: Revert to an earlier version.

```
metadata = {"revertToVersionId": "1"}  
  
self.update_image_set_metadata(  
    data_store_id, image_set_id, version_id, metadata, force  
)
```

- For API details, see [UpdateImageSetMetadata](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Copying an image set

Use the CopyImageSet action to copy an [image set](#) in HealthImaging. You use this asynchronous process to copy the contents of an image set into a new or existing image set. You can copy into a *new* image set to split an image set, as well as to create a separate copy. You can also copy into an *existing* image set to merge two image sets together. For more information, see [CopyImageSet](#) in the *AWS HealthImaging API Reference*.

Note

Keep the following points in mind when using the CopyImageSet action:

- The CopyImageSet action will create a new image set, or a new version of the destinationImageSet. For more information, see [Listing image set versions](#).
- Copy is an asynchronous process. Therefore, the state ([imageSetState](#)) and status ([imageSetWorkflowStatus](#)) response elements are available to let you know what operation is happening on a locked image set. Other write operations cannot be performed on a locked image set.
- CopyImageSet requires SOP Instance UIDs be unique within an image set.
- You can copy subsets of SOP Instances using [copiableAttributes](#). This allows you to pick one or more SOP Instances from the sourceImageSet to copy to the destinationImageSet.
- If the CopyImageSet action is not successful, call GetImageSet and review the [message](#) property. For more information, see [Getting image set properties](#).
- Real-world DICOM imports can result in multiple image sets per DICOM Series. The CopyImageSet action requires sourceImageSet and destinationImageSet to have consistent metadata, unless the optional [force](#) parameter is supplied.
- Set the [force](#) parameter to force the operation, even if there are inconsistent metadata elements between the sourceImageSet and destinationImageSet. In these cases, the Patient, Study, and Series metadata remains unchanged in the destinationImageSet.

To copy an image set

Choose a tab based on your access preference to AWS HealthImaging.

AWS CLI and SDKs

CLI

AWS CLI**Example 1: To copy an image set without a destination.**

The following `copy-image-set` example makes a duplicate copy of an image set without a destination.

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

Output:

```
{
  "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"
}
```

Example 2: To copy an image set with a destination.

The following `copy-image-set` example makes a duplicate copy of an image set with a destination.

```
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"} }'
```

Output:

```
{
  "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"
}
```

Example 3: To copy a subset of instances from a source image set to a destination image set.

The following `copy-image-set` example copies one DICOM instance from the source image set to the destination image set. The `force` parameter is provided to override inconsistencies in the Patient, Study, and Series level attributes.

```
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
```

Output:

```
{
  "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"
}
```

- For API details, see [CopyImageSet](#) in *AWS CLI Command Reference*.

Java**SDK for Java 2.x**

```
/**
 * Copy an AWS HealthImaging image set.
 *
 * @param medicalImagingClient - The AWS HealthImaging client object.
 * @param datastoreId          - The datastore ID.
 * @param imageSetId          - The image set ID.
 * @param latestVersionId     - The version ID.
 * @param destinationImageSetId - The optional destination image set ID,
 * ignored if null.
 * @param destinationVersionId - The optional destination version ID,
 * ignored if null.
 * @param force                - The force flag.
```

```

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

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

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

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

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

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

```

```

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

        CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

```

Utility function to create copiable attributes.

```

/**
 * 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();
}

```

- For API details, see [CopyImageSet](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

Utility function to copy an image set.

```

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);  
    }  
  };  
};
```

Copy an image set without a destination.

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

Copy an image set with a destination.

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

Copy a subset of an image set with a destination and force the copy.

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

- For API details, see [CopyImageSet](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

Utility function to copy an image set.

```
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"]

```

Copy an image set without a destination.

```
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,
)
```

Copy an image set with a destination.

```
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,
)
```

Copy a subset of an image set.

```
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,
            )
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [CopyImageSet](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Deleting an image set

Use the `DeleteImageSet` action to delete an [image set](#) in HealthImaging. The following menus provide a procedure for the AWS Management Console and code examples for the AWS CLI and AWS SDKs. For more information, see [DeleteImageSet](#) in the *AWS HealthImaging API Reference*.

To delete an image set

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens and the **Image sets** tab is selected by default.

3. Choose an image set and choose **Delete**.

The **Delete image set** modal opens.

4. Provide the ID of the image set and choose **Delete image set**.

AWS CLI and SDKs

C++

SDK for C++

```
#!/ Routine which deletes an AWS HealthImaging image set.
/*!
  \param datastoreID: The HealthImaging data store ID.
  \param imageSetID: The image set ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
```

```
*/
bool AwsDoc::Medical_Imaging::deleteImageSet(
    const Aws::String &dataStoreID, const Aws::String &imageSetID,
    const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::DeleteImageSetRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    Aws::MedicalImaging::Model::DeleteImageSetOutcome outcome =
    client.DeleteImageSet(
        request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted image set " << imageSetID
            << " from data store " << dataStoreID << std::endl;
    }
    else {
        std::cerr << "Error deleting image set " << imageSetID << " from data
        store "
            << dataStoreID << ": " <<
            outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see [DeleteImageSet](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To delete an image set

The following `delete-image-set` code example deletes an image set.

```
aws medical-imaging delete-image-set \
```

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

Output:

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

- For API details, see [DeleteImageSet](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static void deleteMedicalImageSet(MedicalImagingClient  
medicalImagingClient,  
    String datastoreId,  
    String imagesetId) {  
    try {  
        DeleteImageSetRequest deleteImageSetRequest =  
DeleteImageSetRequest.builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imagesetId)  
            .build();  
  
        medicalImagingClient.deleteImageSet(deleteImageSetRequest);  
  
        System.out.println("The image set was deleted.");  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- For API details, see [DeleteImageSet](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'DELETING'
  // }
```

```
    return response;
};
```

- For API details, see [DeleteImageSet](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [DeleteImageSet](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Tagging resources with AWS HealthImaging

You can assign metadata to HealthImaging resources ([data stores](#) and [image sets](#)) in the form of tags. Each tag is a label consisting of a user-defined key and value. Tags help you manage, identify, organize, search for, and filter resources.

Important

Do not store protected health information (PHI), personally identifiable information (PII), or other confidential or sensitive information in tags. Tags are not intended to be used for private or sensitive data.

The following topics describe how to use HealthImaging tagging operations using the AWS Management Console, AWS CLI, and AWS SDKs. For more information, see [Tagging your AWS resources](#) in the *AWS General Reference Guide*.

Topics

- [Tagging a resource](#)
- [Listing tags for a resource](#)
- [Untagging a resource](#)

Tagging a resource

Use the [TagResource](#) action to tag [data stores](#) and [image sets](#) in AWS HealthImaging. The following code examples describe how to use the TagResource action with the AWS Management Console, AWS CLI, and AWS SDKs. For more information, see [Tagging your AWS resources](#) in the *AWS General Reference Guide*.

To tag a resource

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens.

3. Choose the **Details** tab.
4. Under the **Tags** section, choose **Manage tags**.

The **Manage tags** page opens.

5. Choose **Add new tag**.
6. Enter a **Key** and **Value** (optional).
7. Choose **Save changes**.

AWS CLI and SDKs

CLI

AWS CLI

Example 1: To tag a data store

The following tag-resource code examples tags a data store.

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

This command produces no output.

Example 2: To tag an image set

The following tag-resource code examples tags an image set.

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

This command produces no output.

- For API details, see [TagResource](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

        medicalImagingClient.tagResource(tagResourceRequest);

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

- For API details, see [TagResource](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
store or image set.
```

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

  return response;
};
```

- For API details, see [TagResource](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [TagResource](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Listing tags for a resource

Use the [ListTagsForResource](#) action to list tags for [data stores](#) and [image sets](#) in AWS HealthImaging. The following code examples describe how to use the `ListTagsForResource` action with the AWS Management Console, AWS CLI, and AWS SDKs. For more information, see [Tagging your AWS resources](#) in the *AWS General Reference Guide*.

To list tags for a resource

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).
2. Choose a data store.

The **Data store details** page opens.

3. Choose the **Details** tab.

Under the **Tags** section, all data store tags are listed.

AWS CLI and SDKs

CLI

AWS CLI

Example 1: To list resource tags for a data store

The following `list-tags-for-resource` code example lists tags for a data store.

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

Output:

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

```
}  
}
```

Example 2: To list resource tags for an image set

The following `list-tags-for-resource` code example lists tags for an image set.

```
aws medical-imaging list-tags-for-resource \  
  --resource-arn "arn:aws:medical-imaging:us-  
east-1:123456789012:datastore/1234567890123456789012/  
imageset/18f88ac7870584f58d56256646b4d92b"
```

Output:

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

- For API details, see [ListTagsForResource](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static ListTagsForResourceResponse  
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,  
    String resourceArn) {  
    try {  
        ListTagsForResourceRequest listTagsForResourceRequest =  
ListTagsForResourceRequest.builder()  
            .resourceArn(resourceArn)  
            .build();  
  
        return  
medicalImagingClient.listTagsForResource(listTagsForResourceRequest);  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

```
    return null;
  }
```

- For API details, see [ListTagsForResource](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 * store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
```

```
// }  
  
    return response;  
};
```

- For API details, see [ListTagsForResource](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

```
class MedicalImagingWrapper:  
    def __init__(self, health_imaging_client):  
        self.health_imaging_client = health_imaging_client  
  
    def list_tags_for_resource(self, resource_arn):  
        """  
        List the tags for a resource.  
  
        :param resource_arn: The ARN of the resource.  
        :return: The list of tags.  
        """  
        try:  
            tags = self.health_imaging_client.list_tags_for_resource(  
                resourceArn=resource_arn  
            )  
        except ClientError as err:  
            logger.error(  
                "Couldn't list tags for resource. Here's why: %s: %s",  
                err.response["Error"]["Code"],  
                err.response["Error"]["Message"],  
            )  
            raise  
        else:
```

```
return tags["tags"]
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListTagsForResource](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Untagging a resource

Use the [UntagResource](#) action to untag [data stores](#) and [image sets](#) in AWS HealthImaging. The following code examples describe how to use the `UntagResource` action with the AWS Management Console, AWS CLI, and AWS SDKs. For more information, see [Tagging your AWS resources](#) in the *AWS General Reference Guide*.

To untag a resource

Choose a menu based on your access preference to AWS HealthImaging.

AWS Console

1. Open the HealthImaging console [Data stores page](#).

2. Choose a data store.

The **Data store details** page opens.

3. Choose the **Details** tab.
4. Under the **Tags** section, choose **Manage tags**.

The **Manage tags** page opens.

5. Choose **Remove** next to the tag you want to remove.
6. Choose **Save changes**.

AWS CLI and SDKs

CLI

AWS CLI

Example 1: To untag a data store

The following `untag-resource` code example untags a data store.

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

This command produces no output.

Example 2: To untag an image set

The following `untag-resource` code example untags an image set.

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

This command produces no output.

- For API details, see [UntagResource](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

        medicalImagingClient.untagResource(untagResourceRequest);

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

- For API details, see [UntagResource](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

  return response;
};
```

- For API details, see [UntagResource](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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


The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [UntagResource](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

 Example availability

Can't find what you need? Request a code example using the **Provide feedback** link at the bottom of this page.

Code examples for HealthImaging using AWS SDKs

The following code examples show how to use HealthImaging with an AWS software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello HealthImaging

The following code examples show how to get started using HealthImaging.

C++

SDK for C++

Code for the CMakeLists.txt CMake file.

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

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

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

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

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

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

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

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

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

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

add_executable(${PROJECT_NAME}
  hello_health_imaging.cpp)

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

Code for the `hello_health_imaging.cpp` source file.

```
#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;
}

```

- For API details, see [ListDatastores](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

```
import {
```

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

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

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

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

- For API details, see [ListDatastores](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

logger = logging.getLogger(__name__)

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

```
and config files.

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

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

- For API details, see [ListDatastores](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Code examples

- [Basic examples for HealthImaging using AWS SDKs](#)
 - [Hello HealthImaging](#)
 - [Actions for HealthImaging using AWS SDKs](#)
 - [Use CopyImageSet with an AWS SDK or CLI](#)

- [Use CreateDatastore with an AWS SDK or CLI](#)
 - [Use DeleteDatastore with an AWS SDK or CLI](#)
 - [Use DeleteImageSet with an AWS SDK or CLI](#)
 - [Use GetDICOMImportJob with an AWS SDK or CLI](#)
 - [Use GetDatastore with an AWS SDK or CLI](#)
 - [Use GetImageFrame with an AWS SDK or CLI](#)
 - [Use GetImageSet with an AWS SDK or CLI](#)
 - [Use GetImageSetMetadata with an AWS SDK or CLI](#)
 - [Use ListDICOMImportJobs with an AWS SDK or CLI](#)
 - [Use ListDatastores with an AWS SDK or CLI](#)
 - [Use ListImageSetVersions with an AWS SDK or CLI](#)
 - [Use ListTagsForResource with an AWS SDK or CLI](#)
 - [Use SearchImageSets with an AWS SDK or CLI](#)
 - [Use StartDICOMImportJob with an AWS SDK or CLI](#)
 - [Use TagResource with an AWS SDK or CLI](#)
 - [Use UntagResource with an AWS SDK or CLI](#)
 - [Use UpdateImageSetMetadata with an AWS SDK or CLI](#)
- [Scenarios for HealthImaging using AWS SDKs](#)
 - [Get started with HealthImaging image sets and image frames using an AWS SDK](#)
 - [Tagging a HealthImaging data store using an AWS SDK](#)
 - [Tagging a HealthImaging image set using an AWS SDK](#)

Basic examples for HealthImaging using AWS SDKs

The following code examples show how to use the basics of AWS HealthImaging with AWS SDKs.

Examples

- [Hello HealthImaging](#)
- [Actions for HealthImaging using AWS SDKs](#)
 - [Use CopyImageSet with an AWS SDK or CLI](#)

- [Use DeleteDatastore with an AWS SDK or CLI](#)
- [Use DeleteImageSet with an AWS SDK or CLI](#)
- [Use GetDICOMImportJob with an AWS SDK or CLI](#)
- [Use GetDatastore with an AWS SDK or CLI](#)
- [Use GetImageFrame with an AWS SDK or CLI](#)
- [Use GetImageSet with an AWS SDK or CLI](#)
- [Use GetImageSetMetadata with an AWS SDK or CLI](#)
- [Use ListDICOMImportJobs with an AWS SDK or CLI](#)
- [Use ListDatastores with an AWS SDK or CLI](#)
- [Use ListImageSetVersions with an AWS SDK or CLI](#)
- [Use ListTagsForResource with an AWS SDK or CLI](#)
- [Use SearchImageSets with an AWS SDK or CLI](#)
- [Use StartDICOMImportJob with an AWS SDK or CLI](#)
- [Use TagResource with an AWS SDK or CLI](#)
- [Use UntagResource with an AWS SDK or CLI](#)
- [Use UpdateImageSetMetadata with an AWS SDK or CLI](#)

Hello HealthImaging

The following code examples show how to get started using HealthImaging.

C++

SDK for C++

Code for the CMakeLists.txt CMake file.

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

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

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

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

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

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

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

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

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

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

add_executable(${PROJECT_NAME}
  hello_health_imaging.cpp)

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

Code for the `hello_health_imaging.cpp` source file.

```
#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;
}
```

- For API details, see [ListDatastores](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

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

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

- For API details, see [ListDatastores](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

logger = logging.getLogger(__name__)
```

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

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

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

- For API details, see [ListDatastores](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Actions for HealthImaging using AWS SDKs

The following code examples demonstrate how to perform individual HealthImaging actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the HealthImaging API and are code excerpts from larger programs that must be run in context. You can see actions in context in [Scenarios for HealthImaging using AWS SDKs](#).

The following examples include only the most commonly used actions. For a complete list, see the [AWS HealthImaging API Reference](#).

Examples

- [Use CopyImageSet with an AWS SDK or CLI](#)
- [Use CreateDatastore with an AWS SDK or CLI](#)
- [Use DeleteDatastore with an AWS SDK or CLI](#)
- [Use DeleteImageSet with an AWS SDK or CLI](#)
- [Use GetDICOMImportJob with an AWS SDK or CLI](#)
- [Use GetDatastore with an AWS SDK or CLI](#)
- [Use GetImageFrame with an AWS SDK or CLI](#)
- [Use GetImageSet with an AWS SDK or CLI](#)
- [Use GetImageSetMetadata with an AWS SDK or CLI](#)
- [Use ListDICOMImportJobs with an AWS SDK or CLI](#)
- [Use ListDatastores with an AWS SDK or CLI](#)
- [Use ListImageSetVersions with an AWS SDK or CLI](#)
- [Use ListTagsForResource with an AWS SDK or CLI](#)
- [Use SearchImageSets with an AWS SDK or CLI](#)
- [Use StartDICOMImportJob with an AWS SDK or CLI](#)
- [Use TagResource with an AWS SDK or CLI](#)
- [Use UntagResource with an AWS SDK or CLI](#)

- [Use UpdateImageSetMetadata with an AWS SDK or CLI](#)

Use CopyImageSet with an AWS SDK or CLI

The following code examples show how to use CopyImageSet.

CLI

AWS CLI

Example 1: To copy an image set without a destination.

The following copy-image-set example makes a duplicate copy of an image set without a destination.

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

Output:

```
{  
  "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"  
}
```

Example 2: To copy an image set with a destination.

The following `copy-image-set` example makes a duplicate copy of an image set with a destination.

```
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"} }'
```

Output:

```
{
  "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"
}
```

Example 3: To copy a subset of instances from a source image set to a destination image set.

The following `copy-image-set` example copies one DICOM instance from the source image set to the destination image set. The `force` parameter is provided to override inconsistencies in the Patient, Study, and Series level attributes.

```
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
```

Output:

```
{
  "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"
}
```

- For API details, see [CopyImageSet](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
/**
```

```

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

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

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

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

        CopyImageSetInformation.Builder copyImageSetBuilder =
            CopyImageSetInformation.builder()

```

```

        .sourceImageSet(copySourceImageSetInformation.build());

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

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

        CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

```

Utility function to create copiable attributes.

```

/**
 * 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();
}

```

- For API details, see [CopyImageSet](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

Utility function to copy an image set.

```
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/xxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
//         imageSetId: 'xxxxxxxxxxxxxxxxxxx',
//         imageSetState: 'LOCKED',
//         imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//         latestVersionId: '4',
//         updatedAt: 2023-09-27T19:46:21.824Z
//     }
// }
return response;
} catch (err) {
    console.error(err);
}
};
```

Copy an image set without a destination.

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


Copy an image set with a destination.

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

Copy a subset of an image set with a destination and force the copy.

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

- For API details, see [CopyImageSet](#) in *AWS SDK for JavaScript API Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

Utility function to copy an image set.

```
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"]

```

Copy an image set without a destination.

```

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,
)

```

Copy an image set with a destination.

```

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,
    )

```

Copy a subset of an image set.

```

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,
)

```

The following code instantiates the `MedicalImagingWrapper` object.

```

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

```

- For API details, see [CopyImageSet](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use CreateDatastore with an AWS SDK or CLI

The following code examples show how to use CreateDatastore.

Bash

AWS CLI with Bash script

```
#####
# 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
    fi
}
```

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

echo "$response"

return 0
}
```

- For API details, see [CreateDatastore](#) in *AWS CLI Command Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To create a data store

The following `create-datastore` code example creates a data store with the name `my-datastore`.

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

Output:

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

- For API details, see [CreateDatastore](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

    return "";
}
```

- For API details, see [CreateDatastore](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
```

```
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- For API details, see [CreateDatastore](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [CreateDatastore](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteDatastore` with an AWS SDK or CLI

The following code examples show how to use `DeleteDatastore`.

Bash

AWS CLI with Bash script

```
#####
# 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
        esac
    done
}
```

```
    ;;
    \?)
    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
}
```

- For API details, see [DeleteDatastore](#) in *AWS CLI Command Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To delete a data store

The following delete-datastore code example deletes a data store.

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

Output:

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

- For API details, see [DeleteDatastore](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

- For API details, see [DeleteDatastore](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

  return response;
};
```

- For API details, see [DeleteDatastore](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [DeleteDatastore](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DeleteImageSet with an AWS SDK or CLI

The following code examples show how to use DeleteImageSet.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with image sets and image frames](#)

C++

SDK for C++

```
//! Routine which deletes an AWS HealthImaging image set.
/*!
 \param datastoreID: The HealthImaging data store ID.
 \param imageSetID: The image set ID.
 \param clientConfig: Aws client configuration.
 \return bool: Function succeeded.
 */
bool AwsDoc::Medical_Imaging::deleteImageSet(
    const Aws::String &dataStoreID, const Aws::String &imageSetID,
    const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::DeleteImageSetRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    Aws::MedicalImaging::Model::DeleteImageSetOutcome outcome =
    client.DeleteImageSet(
        request);
    if (outcome.IsSuccess()) {
```

```

        std::cout << "Successfully deleted image set " << imageSetID
                << " from data store " << dataStoreID << std::endl;
    }
    else {
        std::cerr << "Error deleting image set " << imageSetID << " from data
store "
                << dataStoreID << ": " <<
                outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

```

- For API details, see [DeleteImageSet](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To delete an image set

The following `delete-image-set` code example deletes an image set.

```

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

```

Output:

```

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

```

- For API details, see [DeleteImageSet](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
    String datastoreId,
    String imagesetId) {
    try {
        DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
            .datastoreId(datastoreId)
            .imageSetId(imagesetId)
            .build();

        medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- For API details, see [DeleteImageSet](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

```
/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- For API details, see [DeleteImageSet](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [DeleteImageSet](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetDICOMImportJob with an AWS SDK or CLI

The following code examples show how to use GetDICOMImportJob.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with image sets and image frames](#)

C++

SDK for C++

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

```

        request);
    if (!outcome.IsSuccess()) {
        std::cerr << "GetDICOMImportJob error: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome;
}

```

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To get a dicom import job's properties

The following `get-dicom-import-job` code example gets a dicom import job's properties.

```

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

```

Output:

```

{
  "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/"
    }
}
```

- For API details, see [GetDICOMImportJob](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

    try {
        GetDicomImportJobRequest getDicomImportJobRequest =
        GetDicomImportJobRequest.builder()
            .datastoreId(datastoreId)
            .jobId(jobId)
            .build();

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

    return null;
}
```

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

```

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

```

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

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

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

```

```

        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return job["jobProperties"]

```

The following code instantiates the `MedicalImagingWrapper` object.

```

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

```

- For API details, see [GetDICOMImportJob](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetDatastore with an AWS SDK or CLI

The following code examples show how to use `GetDatastore`.

Bash

AWS CLI with Bash script

```

#####
# 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
}
```

- For API details, see [GetDatastore](#) in *AWS CLI Command Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To get a data store's properties

The following `get-datastore` code example gets a data store's properties.

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

Output:

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

- For API details, see [GetDatastore](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static DatastoreProperties  
getMedicalImageDatastore(MedicalImagingClient medicalImagingClient,  
    String datastoreID) {  
    try {  
        GetDatastoreRequest datastoreRequest = GetDatastoreRequest.builder()  
            .datastoreId(datastoreID)  
            .build();  
        GetDatastoreResponse response =  
            medicalImagingClient.getDatastore(datastoreRequest);  
        return response.datastoreProperties();  
    }  
}
```

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

    return null;
}
```

- For API details, see [GetDatastore](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
    const response = await medicalImagingClient.send(
        new GetDatastoreCommand({ datastoreId: datastoreID }),
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 200,
    //     requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
    //     extendedRequestId: undefined,
    //     cfId: undefined,
    //     attempts: 1,
    //     totalRetryDelay: 0
    //   },
    //   datastoreProperties: {
```

```

//      createdAt: 2023-08-04T18:50:36.239Z,
//      datastoreArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxx:datastore/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      datastoreName: 'my_datastore',
//      datastoreStatus: 'ACTIVE',
//      updatedAt: 2023-08-04T18:50:36.239Z
//    }
// }
return response.datastoreProperties;
};

```

- For API details, see [GetDatastore](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

```

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

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

        :param datastore_id: The ID of the data store.
        :return: The data store properties.
        """
        try:
            data_store = self.health_imaging_client.get_datastore(
                datastoreId=datastore_id
            )
        except ClientError as err:

```



```
        logger.error(
            "Couldn't get data store %s. Here's why: %s: %s",
            id,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return data_store["datastoreProperties"]
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetDatastore](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `GetImageFrame` with an AWS SDK or CLI

The following code examples show how to use `GetImageFrame`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with image sets and image frames](#)

C++

SDK for C++

```

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

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

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

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

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

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

```

```

    }

    return outcome.IsSuccess();
}

```

- For API details, see [GetImageFrame](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

To get image set pixel data

The following `get-image-frame` code example gets an image frame.

```

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

```

Note: This code example does not include output because the `GetImageFrame` action returns a stream of pixel data to the `imageframe.jpg` file. For information about decoding and viewing image frames, see HTJ2K decoding libraries.

- For API details, see [GetImageFrame](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```

    public static void getMedicalImageSetFrame(MedicalImagingClient
        medicalImagingClient,

```

```
        String destinationPath,
        String datastoreId,
        String imagesetId,
        String imageFrameId) {

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


        .imageFrameInformation(ImageFrameInformation.builder()

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

        medicalImagingClient.getImageFrame(getImageSetMetadataRequest,
        FileSystems.getDefault().getPath(destinationPath));

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

- For API details, see [GetImageFrame](#) in *AWS SDK for Java 2.x API Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

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

```
};
```

- For API details, see [GetImageFrame](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

        :param file_path_to_write: The path to write the image frame's HTJ2K
        encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.health_imaging_client.get_image_frame(
                datastoreId=datastore_id,
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)
```

```
except ClientError as err:
    logger.error(
        "Couldn't get image frame. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetImageFrame](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `GetImageSet` with an AWS SDK or CLI

The following code examples show how to use `GetImageSet`.

CLI

AWS CLI

To get image set properties

The following `get-image-set` code example gets the properties for an image set.

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

```
--datastore-id 12345678901234567890123456789012 \  
--image-set-id 18f88ac7870584f58d56256646b4d92b \  
--version-id 1
```

Output:

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

- For API details, see [GetImageSet](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static GetImageSetResponse getMedicalImageSet(MedicalImagingClient  
medicalImagingClient,  
    String datastoreId,  
    String imagesetId,  
    String versionId) {  
    try {  
        GetImageSetRequest.Builder getImageSetRequestBuilder =  
GetImageSetRequest.builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imagesetId);  
  
        if (versionId != null) {  
            getImageSetRequestBuilder =  
getImageSetRequestBuilder.versionId(versionId);  
        }  
  
        return  
medicalImagingClient.getImageSet(getImageSetRequestBuilder.build());  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
    }  
}
```



```
        System.exit(1);
    }

    return null;
}
```

- For API details, see [GetImageSet](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "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);
  // {
```



```
"""
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
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetImageSet](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetImageSetMetadata with an AWS SDK or CLI

The following code examples show how to use GetImageSetMetadata.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with image sets and image frames](#)

C++

SDK for C++

Utility function to get image set metadata.

```
//! Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The HealthImaging image set ID.
  \param versionID: The HealthImaging image set version ID, ignored if empty.
  \param outputPath: The path where the metadata will be stored as gzipped
  json.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
&outputFilePath,
```

```

                                const
    Aws::Client::ClientConfiguration &clientConfig) {
        Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
        request.SetDatastoreId(dataStoreID);
        request.SetImageSetId(imageSetID);
        if (!versionID.empty()) {
            request.SetVersionId(versionID);
        }
        Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
        Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
    client.GetImageSetMetadata(
        request);
        if (outcome.IsSuccess()) {
            std::ofstream file(outputFilePath, std::ios::binary);
            auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
            file << metadata.rdbuf();
        }
        else {
            std::cerr << "Failed to get image set metadata: "
                << outcome.GetError().GetMessage() << std::endl;
        }

        return outcome.IsSuccess();
    }

```

Get image set metadata without version.

```

        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;
        }

```

Get image set metadata with version.

```

        if (AwsDoc::Medical_Imaging::getImageSetMetadata(dataStoreID, imageSetID,
versionID, outputFilePath, clientConfig))
        {

```

```
        std::cout << "Successfully retrieved image set metadata." <<
std::endl;
        std::cout << "Metadata stored in: " << outputPath << std::endl;
    }
}
```

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI

Example 1: To get image set metadata without version

The following `get-image-set-metadata` code example gets metadata for an image set without specifying a version.

Note: `outfile` is a required parameter

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

The returned metadata is compressed with gzip and stored in the `studymetadata.json.gz` file. To view the contents of the returned JSON object, you must first decompress it.

Output:

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

Example 2: To get image set metadata with version

The following `get-image-set-metadata` code example gets metadata for an image set with a specified version.

Note: `outfile` is a required parameter

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

The returned metadata is compressed with gzip and stored in the `studymetadata.json.gz` file. To view the contents of the returned JSON object, you must first decompress it.

Output:

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

- For API details, see [GetImageSetMetadata](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static void getMedicalImageSetMetadata(MedicalImagingClient  
medicalImagingClient,  
    String destinationPath,  
    String datastoreId,  
    String imagesetId,  
    String versionId) {  
  
    try {  
        GetImageSetMetadataRequest.Builder getImageSetMetadataRequestBuilder  
= GetImageSetMetadataRequest.builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imagesetId);  
  
        if (versionId != null) {
```

```

        getImageSetMetadataRequestBuilder =
getImageSetMetadataRequestBuilder.versionId(versionId);
    }

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

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

```

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

Utility function to get image set metadata.

```

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;
};
```

Get image set metadata without version.

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

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

Get image set metadata with version.

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

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

Utility function to get image set metadata.

```
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
```

Get image set metadata without version.

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

Get image set metadata with version.

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [GetImageSetMetadata](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `ListDICOMImportJobs` with an AWS SDK or CLI

The following code examples show how to use `ListDICOMImportJobs`.

CLI

AWS CLI

To list dicom import jobs

The following `list-dicom-import-jobs` code example lists dicom import jobs.

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

Output:

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

- For API details, see [ListDICOMImportJobs](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static List<DICOMImportJobSummary>  
listDicomImportJobs(MedicalImagingClient medicalImagingClient,  
                    String datastoreId) {  
  
    try {  
        ListDicomImportJobsRequest listDicomImportJobsRequest =  
ListDicomImportJobsRequest.builder()  
            .medicalImagingClient(medicalImagingClient)  
            .datastoreId(datastoreId)  
            .build();  
        return medicalImagingClient.listDicomImportJobs(listDicomImportJobsRequest).getJobSummaries();  
    }  
}
```

```

        .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<>();
}

```

- For API details, see [ListDICOMImportJobs](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

```

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


/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
    datastoreId = "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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',  
//       endedAt: 2023-09-22T14:49:51.351Z,  
//       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',  
//       jobName: 'test-1',  
//       jobStatus: 'COMPLETED',  
//       submittedAt: 2023-09-22T14:48:45.767Z  
//     }  
//   ]  
  
  return jobSummaries;  
};
```

- For API details, see [ListDICOMImportJobs](#) in *AWS SDK for JavaScript API Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListDICOMImportJobs](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ListDatastores with an AWS SDK or CLI

The following code examples show how to use ListDatastores.

Bash

AWS CLI with Bash script

```
#####
# 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 getopts "h" option; do
    case "${option}" in
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

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

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

echo "$response"

return 0
}
```

- For API details, see [ListDatastores](#) in *AWS CLI Command Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI**To list data stores**

The following `list-datastores` code example lists available data stores.

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

Output:

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

- For API details, see [ListDatastores](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static List<DatastoreSummary>
listMedicalImagingDatastores(MedicalImagingClient medicalImagingClient) {
```

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

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

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

    return null;
}
```

- For API details, see [ListDatastores](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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


Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListDatastores](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `ListImageSetVersions` with an AWS SDK or CLI

The following code examples show how to use `ListImageSetVersions`.

CLI

AWS CLI

To list image set versions

The following `list-image-set-versions` code example lists the version history for an image set.

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

Output:

```
{
  "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
    }
]
}

```

- For API details, see [ListImageSetVersions](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```

public static List<ImageSetProperties>
listMedicalImageSetVersions(MedicalImagingClient medicalImagingClient,
    String datastoreId,

```



```

        String imagesetId) {
    try {
        ListImageSetVersionsRequest getImageSetRequest =
ListImageSetVersionsRequest.builder()
        .datastoreId(datastoreId)
        .imageSetId(imagesetId)
        .build();

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

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

    return null;
}

```

- For API details, see [ListImageSetVersions](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

```

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

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

```

```
*/
export const listImageSetVersions = async (
  datastoreId = "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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       imageSetState: 'ACTIVE',
  //       versionId: '1'
  //     }
  //   ]
  // }
  return imageSetPropertiesList;
};
```

- For API details, see [ListImageSetVersions](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

```
    )
    raise
else:
    return image_set_properties_list
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListImageSetVersions](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `ListTagsForResource` with an AWS SDK or CLI

The following code examples show how to use `ListTagsForResource`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Tagging a data store](#)
- [Tagging an image set](#)

CLI

AWS CLI

Example 1: To list resource tags for a data store

The following `list-tags-for-resource` code example lists tags for a data store.

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

Output:

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

Example 2: To list resource tags for an image set

The following `list-tags-for-resource` code example lists tags for an image set.

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

Output:

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

- For API details, see [ListTagsForResource](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static ListTagsForResourceResponse  
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,  
    String resourceArn) {
```

```
    try {
        ListTagsForResourceRequest listTagsForResourceRequest =
ListTagsForResourceRequest.builder()
        .resourceArn(resourceArn)
        .build();

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

    return null;
}
```

- For API details, see [ListTagsForResource](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

```
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   tags: { Deployment: 'Development' }
// }

return response;
};
```

- For API details, see [ListTagsForResource](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

        :param resource_arn: The ARN of the resource.
        :return: The list of tags.
        """
```

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [ListTagsForResource](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `SearchImageSets` with an AWS SDK or CLI

The following code examples show how to use `SearchImageSets`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with image sets and image frames](#)

C++

SDK for C++

The utility function for searching image sets.

```
//! 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;
}

```

Use case #1: EQUAL operator.

```

    Aws::Vector<Aws::String> imageIDsForPatientID;
    Aws::MedicalImaging::Model::SearchCriteria searchCriteriaEqualsPatientID;
    Aws::Vector<Aws::MedicalImaging::Model::SearchFilter>
patientIDSearchFilters = {

    Aws::MedicalImaging::Model::SearchFilter().WithOperator(Aws::MedicalImaging::Model::Oper

    .WithValues({Aws::MedicalImaging::Model::SearchByAttributeValue().WithDICOMPatientId(pat
        });

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

searchCriteriaEqualsPatientID,

imageIDsForPatientID,

                                                                    clientConfig);

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

```

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```

    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;
        }
    }
}

```

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```

    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;
        }
    }
}

```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```

    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;
            }
        }
    }
}

```

- For API details, see [SearchImageSets](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI**Example 1: To search image sets with an EQUAL operator**

The following `search-image-sets` code example uses the EQUAL operator to search image sets based on a specific value.

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

Contents of `search-criteria.json`

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

Output:

```
{  
  "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"
  ]
}

```

Example 2: To search image sets with a BETWEEN operator using DICOMStudyDate and DICOMStudyTime

The following search-image-sets code example searches for image sets with DICOM Studies generated between January 1, 1990 (12:00 AM) and January 1, 2023 (12:00 AM).

Note: DICOMStudyTime is optional. If it is not present, 12:00 AM (start of the day) is the time value for the dates provided for filtering.

```

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

```

Contents of search-criteria.json

```

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

```

Output:

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

Example 3: To search image sets with a BETWEEN operator using createdAt (time studies were previously persisted)

The following search-image-sets code example searches for image sets with DICOM Studies persisted in HealthImaging between the time ranges in UTC time zone.

Note: Provide createdAt in example format ("1985-04-12T23:20:50.52Z").

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

Contents of search-criteria.json

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



```

    },
    {
      "createdAt": "2022-04-12T23:20:50.52Z"
    }
  ],
  "operator": "BETWEEN"
}]
}

```

Output:

```

{
  "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"
  }
]
}

```

Example 4: To search image sets with an EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field

The following search-image-sets code example searches for image sets with an EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

Note: Provide updatedAt in example format ("1985-04-12T23:20:50.52Z").

```
aws medical-imaging search-image-sets \
```

```
--datastore-id 12345678901234567890123456789012 \  
--search-criteria file://search-criteria.json
```

Contents of 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"  
  }  
}
```

Output:

```
{  
  "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"
  }]
}

```

- For API details, see [SearchImageSets](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

The utility function for searching image sets.

```

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;
}

```

Use case #1: EQUAL operator.

```

        List<SearchFilter> searchFilters =
Collections.singletonList(SearchFilter.builder()
        .operator(Operator.EQUAL)
        .values(SearchByAttributeValue.builder()
        .dicomPatientId(patientId)
        .build())
        .build());

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

List<ImageSetsMetadataSummary> imageSetsMetadataSummaries =
searchMedicalImagingImageSets(
        medicalImagingClient,
        datastoreId, searchCriteria);
if (imageSetsMetadataSummaries != null) {
    System.out.println("The image sets for patient " + patientId + " are:
\n"
        + imageSetsMetadataSummaries);
    System.out.println();
}

```

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```

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

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

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

```

```

        .build())
        .build());

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

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

```

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```

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();
    }

```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```

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();
}

```

```
}
```

- For API details, see [SearchImageSets](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

The utility function for searching image sets.

```
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;
};
```

Use case #1: EQUAL operator.

```
const datastoreId = "12345678901234567890123456789012";

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



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

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```
const datastoreId = "12345678901234567890123456789012";

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

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

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```
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);
}
```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```
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",
      },
    ],
  };

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

```

    },
  ],
  sort: {
    sortOrder: "ASC",
    sortField: "updatedAt",
  },
};

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

```

- For API details, see [SearchImageSets](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

The utility function for searching image sets.

```

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

```

Use case #1: EQUAL operator.

```

search_filter = {
    "filters": [
        {"operator": "EQUAL", "values": [{"DICOMPatientId": patient_id}]}
    ]
}

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

```

Use case #2: BETWEEN operator using DICOMStudyDate and DICOMStudyTime.

```

search_filter = {
    "filters": [
        {
            "operator": "BETWEEN",
            "values": [
                {
                    "DICOMStudyDateAndTime": {
                        "DICOMStudyDate": "19900101",

```

```

        "DICOMStudyTime": "000000",
    },
    ],
    "DICOMStudyDateAndTime": {
        "DICOMStudyDate": "20230101",
        "DICOMStudyTime": "000000",
    },
    ],
}

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

```

Use case #3: BETWEEN operator using createdAt. Time studies were previously persisted.

```

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}"
)

```

Use case #4: EQUAL operator on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort response in ASC order on updatedAt field.

```

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}")

```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [SearchImageSets](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `StartDICOMImportJob` with an AWS SDK or CLI

The following code examples show how to use `StartDICOMImportJob`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with image sets and image frames](#)

C++

SDK for C++

```
//! Routine which starts a HealthImaging import job.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param inputBucketName: The name of the Amazon S3 bucket containing the DICOM
files.
  \param inputDirectory: The directory in the S3 bucket containing the DICOM
files.
  \param outputBucketName: The name of the S3 bucket for the output.
  \param outputDirectory: The directory in the S3 bucket to store the output.
```

```

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

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

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

    return startDICOMImportJobOutcome.IsSuccess();
}

```

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for C++ API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

CLI

AWS CLI**To start a dicom import job**

The following `start-dicom-import-job` code example starts a dicom import job.

```
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"
```

Output:

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

- For API details, see [StartDICOMImportJob](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
public static String startDicomImportJob(MedicalImagingClient  
medicalImagingClient,  
    String jobName,
```

```

        String datastoreId,
        String dataAccessRoleArn,
        String inputS3Uri,
        String outputS3Uri) {

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

    return "";
}

```

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

```

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

/**

```

```

* @param {string} jobName - The name of the import job.
* @param {string} datastoreId - The ID of the data store.
* @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
that grants permission.
* @param {string} inputS3Uri - The URI of the S3 bucket containing the input
files.
* @param {string} outputS3Uri - The URI of the S3 bucket where the output files
are stored.
*/
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/",
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   jobStatus: 'SUBMITTED',
  //   submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};

```

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

```

```
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return job["jobId"]
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [StartDICOMImportJob](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use TagResource with an AWS SDK or CLI

The following code examples show how to use `TagResource`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Tagging a data store](#)
- [Tagging an image set](#)

CLI

AWS CLI

Example 1: To tag a data store

The following `tag-resource` code examples tags a data store.

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

This command produces no output.

Example 2: To tag an image set

The following `tag-resource` code examples tags an image set.

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

This command produces no output.

- For API details, see [TagResource](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

```
        .build();

        medicalImagingClient.tagResource(tagResourceRequest);

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

- For API details, see [TagResource](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 * store or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 * - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
    resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
    tags = {},
) => {
    const response = await medicalImagingClient.send(
        new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
    );
    console.log(response);
}
```

```
// {
//   '$metadata': {
//     httpStatusCode: 204,
//     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   }
// }

return response;
};
```

- For API details, see [TagResource](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

        :param resource_arn: The ARN of the resource.
        :param tags: The tags to apply.
        """
        try:
            self.health_imaging_client.tag_resource(resourceArn=resource_arn,
            tags=tags)
```



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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [TagResource](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `UntagResource` with an AWS SDK or CLI

The following code examples show how to use `UntagResource`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Tagging a data store](#)
- [Tagging an image set](#)

CLI

AWS CLI

Example 1: To untag a data store

The following `untag-resource` code example untags a data store.

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

This command produces no output.

Example 2: To untag an image set

The following `untag-resource` code example untags an image set.

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

This command produces no output.

- For API details, see [UntagResource](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

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

```
        .tagKeys(tagKeys)
        .build();

    medicalImagingClient.untagResource(untagResourceRequest);

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

- For API details, see [UntagResource](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 * store or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
    resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/
imageset/xxx",
    tagKeys = [],
) => {
    const response = await medicalImagingClient.send(
        new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
    );
    console.log(response);
}
```

```

// {
//   '$metadata': {
//     httpStatusCode: 204,
//     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   }
// }

return response;
};

```

- For API details, see [UntagResource](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

```

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

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

        :param resource_arn: The ARN of the resource.
        :param tag_keys: The tag keys to remove.
        """
        try:
            self.health_imaging_client.untag_resource(
                resourceArn=resource_arn, tagKeys=tag_keys
            )

```

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

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see [UntagResource](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateImageSetMetadata` with an AWS SDK or CLI

The following code examples show how to use `UpdateImageSetMetadata`.

CLI

AWS CLI

Example 1: To insert or update an attribute in image set metadata

The following `update-image-set-metadata` example inserts or updates an attribute in image set metadata.

```
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
```

Contents of metadata-updates.json

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

Output:

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

Example 2: To remove an attribute from image set metadata

The following update-image-set-metadata example removes an attribute from image set metadata.

```
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
```

Contents of metadata-updates.json

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

Output:

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

Example 3: To remove an instance from image set metadata

The following `update-image-set-metadata` example removes an instance from image set metadata.

```
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
```

Contents of `metadata-updates.json`

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

Output:

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

Example 4: To revert an image set to a previous version

The following `update-image-set-metadata` example shows how to revert an image set to a prior version. `CopyImageSet` and `UpdateImageSetMetadata` actions create new versions of image sets.

```
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"}'
```

Output:

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

Example 5: To add a private DICOM data element to an instance

The following `update-image-set-metadata` example shows how to add a private element to a specified instance within an image set. The DICOM standard permits private data elements for communication of information that cannot be contained in standard

data elements. You can create, update, and delete private data elements with the `UpdateImageSetMetadata` action.

```
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
```

Contents of `metadata-updates.json`

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\": {\"001910F9\": \"97\"}, \"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}}}"
  }
}
```

Output:

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

Example 6: To update a private DICOM data element to an instance

The following `update-image-set-metadata` example shows how to update the value of a private data element belonging to an instance within an image set.

```
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

```

Contents of metadata-updates.json

```

{
  "DICOMUpdates": {
    "updatableAttributes": "{\\"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\\": {\\"DICOM\\":
{\\"00091001\\": \\"GE_GENESIS_DD\\"}}}}}}}"
  }
}

```

Output:

```

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

```

Example 7: To update a SOPInstanceUID with the force parameter

The following update-image-set-metadata example shows how to update a SOPInstanceUID, using the force parameter to override the DICOM metadata constraints.

```

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

```

Contents of 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\\"}}}}}}}"
  }
}
```

Output:

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

- For API details, see [UpdateImageSetMetadata](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

```
/**
 * Update the metadata of an AWS HealthImaging image set.
 *
 * @param medicalImagingClient - The AWS HealthImaging client object.
 * @param datastoreId          - The datastore ID.
 * @param imageSetId           - The image set ID.
 * @param versionId            - The version ID.
 * @param metadataUpdates      - A MetadataUpdates object containing the
updates.
 * @param force                 - The force flag.
```

```

    * @throws MedicalImagingException - Base exception for all service
    exceptions thrown by AWS HealthImaging.
    */
    public static void updateMedicalImageSetMetadata(MedicalImagingClient
medicalImagingClient,
                                                    String datastoreId,
                                                    String imageSetId,
                                                    String versionId,
                                                    MetadataUpdates
metadataUpdates,
                                                    boolean force) {
        try {
            UpdateImageSetMetadataRequest updateImageSetMetadataRequest =
UpdateImageSetMetadataRequest
                .builder()
                .datastoreId(datastoreId)
                .imageSetId(imageSetId)
                .latestVersionId(versionId)
                .updateImageSetMetadataUpdates(metadataUpdates)
                .force(force)
                .build();

            UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

```

Use case #1: Insert or update an attribute.

```

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);

```

Use case #2: Remove an attribute.

```

        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);

```

Use case #3: Remove an instance.

```

        final String removeInstance = ""
        {
            "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": {}
                }
            }
        }
    }
}
""";
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);

```

Use case #4: Revert to a previous version.

```

// 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);

```

- For API details, see [UpdateImageSetMetadata](#) in *AWS SDK for Java 2.x API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

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

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

```

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

```

Use case #1: Insert or update an attribute and force the update.

```

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,  
  );
```

Use case #2: Remove an attribute.

```
// 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,  
);
```

Use case #3: Remove an instance.

```
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,
  );
```

Use case #4: Revert to an earlier version.

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

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

- For API details, see [UpdateImageSetMetadata](#) in *AWS SDK for JavaScript API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

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

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

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

The following code instantiates the `MedicalImagingWrapper` object.

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

Use case #1: Insert or update an attribute.

```
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
)
```

Use case #2: Remove an attribute.

```
# 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
)
```

Use case #3: Remove an instance.

```
attributes = """{
    "SchemaVersion": 1.1,
    "Study": {
        "Series": {
            "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                "Instances": {
                    "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                }
            }
        }
    }
}"""
metadata = {"DICOMUpdates": {"removableAttributes": attributes}}

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

Use case #4: Revert to an earlier version.

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

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

- For API details, see [UpdateImageSetMetadata](#) in *AWS SDK for Python (Boto3) API Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for HealthImaging using AWS SDKs

The following code examples show you how to implement common scenarios in HealthImaging with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within HealthImaging or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- [Get started with HealthImaging image sets and image frames using an AWS SDK](#)
- [Tagging a HealthImaging data store using an AWS SDK](#)
- [Tagging a HealthImaging image set using an AWS SDK](#)

Get started with HealthImaging image sets and image frames using an AWS SDK

The following code examples show how to import DICOM files and download image frames in HealthImaging.

The implementation is structured as a workflow command-line application.

- Set up resources for a DICOM import.
- Import DICOM files into a data store.
- Retrieve the image set IDs for the import job.
- Retrieve the image frame IDs for the image sets.
- Download, decode and verify the image frames.
- Clean up resources.

C++

SDK for C++

Create an AWS CloudFormation stack with the necessary resources.

```
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: ");
    }

```

Copy DICOM files to the Amazon S3 import bucket.

```

std::cout
    << "This workflow uses DICOM files from the National Cancer Institute
Imaging Data\n"
    << "Commons (IDC) Collections." << std::endl;
std::cout << "Here is the link to their website." << std::endl;
std::cout << "https://registry.opendata.aws/nci-imaging-data-commons/" <<
std::endl;
std::cout << "We will use DICOM files stored in an S3 bucket managed by the
IDC."
    << std::endl;
std::cout
    << "First one of the DICOM folders in the IDC collection must be
copied to your\n"
    "input S3 bucket."
    << std::endl;
std::cout << "You have the choice of one of the following "
    << IDC_ImageChoices.size() << " folders to copy." << std::endl;

int index = 1;
for (auto &idcChoice: IDC_ImageChoices) {
    std::cout << index << " - " << idcChoice.mDescription << std::endl;
    index++;
}
int choice = askQuestionForIntRange("Choose DICOM files to import: ", 1, 4);

Aws::String fromDirectory = IDC_ImageChoices[choice - 1].mDirectory;

```



```

    Aws::String inputDirectory = "input";

    std::cout << "The files in the directory '" << fromDirectory << "' in the
    bucket '"
        << IDC_S3_BucketName << "' will be copied " << std::endl;
    std::cout << "to the folder '" << inputDirectory << "/" << fromDirectory
        << "' in the bucket '" << inputBucketName << "'." << std::endl;
    askQuestion("Enter return to start the copy.", alwaysTrueTest);

    if (!AwsDoc::Medical_Imaging::copySeriesBetweenBuckets(
        IDC_S3_BucketName,
        fromDirectory,
        inputBucketName,
        inputDirectory, clientConfiguration)) {
        std::cerr << "This workflow will exit because of an error." << std::endl;
        cleanup(stackName, dataStoreId, clientConfiguration);
        return false;
    }

```

Import the DICOM files to the Amazon S3 data store.

```

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;
}

```

Get image sets created by the DICOM import job.

```
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;
}

```

Get image frame information for image sets.

```

bool AwsDoc::Medical_Imaging::getImageFramesForImageSet(const Aws::String
&dataStoreID,
                                                         const Aws::String
&imageSetID,
                                                         const Aws::String
&outDirectory,
                                                         Aws::Vector<ImageFrameInfo> &imageFrames,
                                                         const
Aws::Client::ClientConfiguration &clientConfiguration) {

```

```

    Aws::String fileName = outDirectory + "/" + imageSetID +
    "_metadata.json.gzip";
    bool result = false;
    if (getImageSetMetadata(dataStoreID, imageSetID, "", // Empty string for
    version ID.
                            fileName, clientConfiguration)) {
    try {
        std::string metadataGZip;
        {
            std::ifstream inFileStream(fileName.c_str(), std::ios::binary);
            if (!inFileStream) {
                throw std::runtime_error("Failed to open file " + fileName);
            }

            std::stringstream stringStream;
            stringStream << inFileStream.rdbuf();
            metadataGZip = stringStream.str();
        }
        std::string metadataJson = gzip::decompress(metadataGZip.data(),
                                                    metadataGZip.size());
        // Use JMESPath to extract the image set IDs.
        // https://jmespath.org/specification.html
        jsoncons::json doc = jsoncons::json::parse(metadataJson);
        std::string jmesPathExpression = "Study.Series.*.Instances[].[*]";
        jsoncons::json instances = jsoncons::jmespath::search(doc,
jmesPathExpression);
        for (auto &instance: instances.array_range()) {
            jmesPathExpression = "DICOM.RescaleSlope";
            std::string rescaleSlope = jsoncons::jmespath::search(instance,
jmesPathExpression).to_string();
            jmesPathExpression = "DICOM.RescaleIntercept";
            std::string rescaleIntercept =
jsoncons::jmespath::search(instance,
jmesPathExpression).to_string();

            jmesPathExpression = "ImageFrames[].[*]";
            jsoncons::json imageFramesJson =
jsoncons::jmespath::search(instance,
jmesPathExpression);

```

```

        for (auto &imageFrame: imageFramesJson.array_range()) {
            ImageFrameInfo imageFrameIDs;
            imageFrameIDs.mImageSetId = imageSetID;
            imageFrameIDs.mImageFrameId = imageFrame.find(
                "ID")->value().as_string();
            imageFrameIDs.mRescaleIntercept = rescaleIntercept;
            imageFrameIDs.mRescaleSlope = rescaleSlope;
            imageFrameIDs.MinPixelValue = imageFrame.find(
                "MinPixelValue")->value().as_string();
            imageFrameIDs.MaxPixelValue = imageFrame.find(
                "MaxPixelValue")->value().as_string();

            jmesPathExpression =
                "max_by(PixelDataChecksumFromBaseToFullResolution, &Width).Checksum";
            jsoncons::json checksumJson =
                jsoncons::jmespath::search(imageFrame,

                jmesPathExpression);
            imageFrameIDs.mFullResolutionChecksum =
                checksumJson.as_integer<uint32_t>();

            imageFrames.emplace_back(imageFrameIDs);
        }
    }

    result = true;
}
catch (const std::exception &e) {
    std::cerr << "getImageFramesForImageSet failed because " << e.what()
        << std::endl;
}
}

return result;
}

//! Routine which gets a HealthImaging image set's metadata.
/*!
    \param dataStoreID: The HealthImaging data store ID.
    \param imageSetID: The HealthImaging image set ID.
    \param versionID: The HealthImaging image set version ID, ignored if empty.
    \param outputPath: The path where the metadata will be stored as gzipped
    json.
    \param clientConfig: Aws client configuration.

```



```

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

```

Download, decode and verify image frames.

```

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;
}

```

Clean up resources.

```

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;
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
 - [DeleteImageSet](#)
 - [GetDICOMImportJob](#)
 - [GetImageFrame](#)
 - [GetImageSetMetadata](#)
 - [SearchImageSets](#)
 - [StartDICOMImportJob](#)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

index.js - Orchestrate steps.

```
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]",
    });
}

```

deploy-steps.js - Deploy resources.

```

import fs from "node:fs/promises";
import path from "node:path";

import {
    CloudFormationClient,
    CreateStackCommand,
    DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
    ScenarioAction,
    ScenarioInput,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

```

```
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
```

```
async (** @type {} *) state => {
  const stackName = state.getStackName;
  const datastoreName = state.getDatastoreName;
  const accountId = state.accountId;

  const command = new CreateStackCommand({
    StackName: stackName,
    TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
    Capabilities: ["CAPABILITY_IAM"],
    Parameters: [
      {
        ParameterKey: "datastoreName",
        ParameterValue: datastoreName,
      },
      {
        ParameterKey: "userAccountID",
        ParameterValue: accountId,
      },
    ],
  });

  const response = await cfnClient.send(command);
  state.stackId = response.StackId;
},
{ skipWhen: (** @type {} *) state => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (** @type {} *) state => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
```

```

        acc[output.OutputKey] = output.OutputValue;
        return acc;
    }, {}));
} else {
    throw new Error(
        `Stack creation failed with status: ${stack.StackStatus}`,
    );
}
});
},
{
    skipWhen: (/** @type {} */ state) => !state.deployStack,
},
);

export const outputState = new ScenarioOutput(
    "outputState",
    (/** @type {} */ state) => {
        /**
         * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
         string }}}
         */
        const { stackOutputs } = state;
        return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
`;
    },
    { skipWhen: (/** @type {} */ state) => !state.deployStack },
);

```

dataset-steps.js - Copy DICOM files.

```

import {
    S3Client,
    CopyObjectCommand,
    ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
    ScenarioAction,

```

```
ScenarioInput,  
ScenarioOutput,  
} from "@aws-doc-sdk-examples/lib/scenario/index.js";  
  
const s3Client = new S3Client({});  
  
const datasetOptions = [  
  {  
    name: "CT of chest (2 images)",  
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",  
  },  
  {  
    name: "CT of pelvis (57 images)",  
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",  
  },  
  {  
    name: "MRI of head (192 images)",  
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",  
  },  
  {  
    name: "MRI of breast (92 images)",  
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",  
  },  
];  
  
/**  
 * @typedef {{ stackOutputs: {  
 *   BucketName: string,  
 *   DatastoreID: string,  
 *   doCopy: boolean  
 * }}} State  
 */  
  
export const selectDataset = new ScenarioInput(  
  "selectDataset",  
  (state) => {  
    if (!state.doCopy) {  
      process.exit(0);  
    }  
    return "Select a DICOM dataset to import:";  
  },  
  {  
    type: "select",  
    choices: datasetOptions,  
  }  
);
```



```
    },
  );

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = "input/";
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey}
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });
  });
```

```
    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js - Start import into datastore.

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

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);
```

```
export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreId,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreId,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);
```

```
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js - Get image set IDs.

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
 [] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
```

```

    Key: key,
  });

  const response = await s3Client.send(command);
  const manifestContent = await response.Body.transformToString();
  state.manifestContent = JSON.parse(manifestContent);
},
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  /** @type {State} */ state) => {
  const imageSetIds =
    state.manifestContent.jobSummary.imageSetsSummary.reduce((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")}`;
);

```

image-frame-steps.js - Get image frame IDs.

```

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;
  },
);

```

verify-steps.js - Verify image frames. The [AWS HealthImaging Pixel Data Verification](#) library was used for verification.

```

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,
            ]
          );
        }
      }
    }
  }
);

```



```
/**
 * @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,
  },
);
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [DeleteImageSet](#)
 - [GetDICOMImportJob](#)
 - [GetImageFrame](#)
 - [GetImageSetMetadata](#)
 - [SearchImageSets](#)
 - [StartDICOMImportJob](#)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

Create an AWS CloudFormation stack with the necessary resources.

```
def deploy(self):
    """
    Deploys prerequisite resources used by the scenario. The resources are
    defined in the associated `setup.yaml` AWS CloudFormation script and are
    deployed
    as a CloudFormation stack, so they can be easily managed and destroyed.
    """

    print("\t\tLet's deploy the stack for resource creation.")
    stack_name = q.ask("\t\tEnter a name for the stack: ", q.non_empty)

    data_store_name = q.ask(
        "\t\tEnter a name for the Health Imaging Data Store: ", q.non_empty
    )

    account_id = boto3.client("sts").get_caller_identity()["Account"]

    with open(
        "../../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml"
    ) as setup_file:
        setup_template = setup_file.read()
    print(f"\t\tCreating {stack_name}.")
    stack = self.cf_resource.create_stack(
        StackName=stack_name,
        TemplateBody=setup_template,
        Capabilities=["CAPABILITY_NAMED_IAM"],
        Parameters=[
            {
                "ParameterKey": "datastoreName",
                "ParameterValue": data_store_name,
            },
            {
                "ParameterKey": "userAccountID",
                "ParameterValue": account_id,
            },
        ],
    )
```

```

print("\t\tWaiting for stack to deploy. This typically takes a minute or
two.")
waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
waiter.wait(StackName=stack.name)
stack.load()
print(f"\t\tStack status: {stack.stack_status}")

outputs_dictionary = {
    output["OutputKey"]: output["OutputValue"] for output in
stack.outputs
}
self.input_bucket_name = outputs_dictionary["BucketName"]
self.output_bucket_name = outputs_dictionary["BucketName"]
self.role_arn = outputs_dictionary["RoleArn"]
self.data_store_id = outputs_dictionary["DatastoreID"]
return stack

```

Copy DICOM files to the Amazon S3 import bucket.

```

def copy_single_object(self, key, source_bucket, target_bucket,
target_directory):
    """
    Copies a single object from a source to a target bucket.

    :param key: The key of the object to copy.
    :param source_bucket: The source bucket for the copy.
    :param target_bucket: The target bucket for the copy.
    :param target_directory: The target directory for the copy.
    """
    new_key = target_directory + "/" + key
    copy_source = {"Bucket": source_bucket, "Key": key}
    self.s3_client.copy_object(
        CopySource=copy_source, Bucket=target_bucket, Key=new_key
    )
    print(f"\n\t\tCopying {key}.")

def copy_images(
    self, source_bucket, source_directory, target_bucket, target_directory
):
    """

```


Copies the images from the source to the target bucket using multiple threads.

```

:param source_bucket: The source bucket for the images.
:param source_directory: Directory within the source bucket.
:param target_bucket: The target bucket for the images.
:param target_directory: Directory within the target bucket.
"""

# Get list of all objects in source bucket.
list_response = self.s3_client.list_objects_v2(
    Bucket=source_bucket, Prefix=source_directory
)
objs = list_response["Contents"]
keys = [obj["Key"] for obj in objs]

# Copy the objects in the bucket.
for key in keys:
    self.copy_single_object(key, source_bucket, target_bucket,
target_directory)

print("\t\tDone copying all objects.")

```

Import the DICOM files to the Amazon S3 data store.

```

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"]
```

Get image sets created by the DICOM import job.

```
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

```

Get image frame information for image sets.

```

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
```

Download, decode and verify image frames.

```
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

```

Clean up resources.

```

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
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [DeleteImageSet](#)
 - [GetDICOMImportJob](#)
 - [GetImageFrame](#)

- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [StartDICOMImportJob](#)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Tagging a HealthImaging data store using an AWS SDK

The following code examples show how to tag a HealthImaging data store.

Java

SDK for Java 2.x

To tag a data store.

```
final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

TagResource.tagMedicalImagingResource(medicalImagingClient,
datastoreArn,
    ImmutableMap.of("Deployment", "Development"));
```

The utility function for tagging a resource.

```
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);
}
}

```

To list tags for a data store.

```

    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());
    }
}

```

The utility function for listing a resource's tags.

```

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;
}

```

To untag a data store.

```

        final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

        UntagResource.untagMedicalImagingResource(medicalImagingClient,
datastoreArn,
            Collections.singletonList("Deployment"));

```

The utility function for untagging a resource.

```

    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);
        }
    }
}

```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
 - [ListTagsForResource](#)

- [TagResource](#)
- [UntagResource](#)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

To tag a data store.

```
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);
}
```

The utility function for tagging a resource.

```
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;
};
```

To list tags for a data store.

```
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);
}
```

The utility function for listing a resource's tags.

```
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;
};

```

To untag a data store.

```

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);
}

```

The utility function for untagging a resource.

```

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;
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

To tag a data store.

```
a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
"Development"})
```

The utility function for tagging a resource.

```
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
```

To list tags for a data store.

```
a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

The utility function for listing a resource's tags.

```
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"]
```

To untag a data store.

```
a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

The utility function for untagging a resource.

```
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
```

The following code instantiates the `MedicalImagingWrapper` object.

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

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Tagging a HealthImaging image set using an AWS SDK

The following code examples show how to tag a HealthImaging image set.

Java

SDK for Java 2.x

To tag an image set.

```
        final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

        TagResource.tagMedicalImagingResource(medicalImagingClient,
imageSetArn,
                                           ImmutableMap.of("Deployment", "Development"));
```

The utility function for tagging a resource.

```
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);
    }
}

```

To list tags for an image set.

```

        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());
        }
    }
}

```

The utility function for listing a resource's tags.

```

    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;
}
```

To untag an image set.

```
        final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

        UntagResource.untagMedicalImagingResource(medicalImagingClient,
imageSetArn,
                Collections.singletonList("Deployment"));
```

The utility function for untagging a resource.

```
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);
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
 - [ListTagsForResource](#)
 - [TagResource](#)

- [UntagResource](#)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

JavaScript

SDK for JavaScript (v3)

To tag an image set.

```
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);
}
```

The utility function for tagging a resource.

```
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;
  };

```

To list tags for an image set.

```

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);
}

```

The utility function for listing a resource's tags.

```

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;
};
```

To untag an image set.

```
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);
}
```


The utility function for untagging a resource.

```
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;
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Python

SDK for Python (Boto3)

To tag an image set.

```
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"})
```

The utility function for tagging a resource.

```
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
```

To list tags for an image set.

```
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)
```

The utility function for listing a resource's tags.

```
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"]
```

To untag an image set.

```

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"])

```

The utility function for untagging a resource.

```

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

```


The following code instantiates the `MedicalImagingWrapper` object.

```

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

```


- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For a complete list of AWS SDK developer guides and code examples, see [Using HealthImaging with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Using DICOMweb with AWS HealthImaging

You can retrieve DICOM objects from AWS HealthImaging using a representation of [DICOMweb](#) APIs, which are web-based services that follow the DICOM standard for medical imaging. This functionality enables you to interoperate with systems that utilize DICOM Part 10 binaries while simultaneously taking advantage of HealthImaging's [cloud native actions](#). The focus of this chapter is how to use HealthImaging's implementation of DICOMweb services to return DICOMweb responses.

Important

HealthImaging import jobs process DICOM instance binaries (. dcm files) and transform them into [image sets](#). Use HealthImaging [cloud native actions](#) (APIs) to manage image sets. Use HealthImaging's representation of DICOMweb services to return DICOMweb responses. The APIs listed in this chapter are built in conformance to the [DICOMweb](#) standard for web-based medical imaging. Because they are representations of DICOMweb services, they are not offered through AWS CLI and AWS SDKs.

Topic

- [Retrieving DICOM data from HealthImaging](#)

Retrieving DICOM data from HealthImaging

AWS HealthImaging offers representations of [DICOMweb WADO-RS](#) services to retrieve a DICOM instance, DICOM instance metadata, and DICOM instance frames (pixel data) from a HealthImaging [data store](#). HealthImaging's DICOMweb WADO-RS services offer flexibility in how you retrieve data stored in HealthImaging and provide interoperability with legacy applications.

Important

HealthImaging import jobs process DICOM instance binaries (. dcm files) and transform them into [image sets](#). Use HealthImaging [cloud native actions](#) (APIs) to manage image sets. Use HealthImaging's representation of DICOMweb services to return DICOMweb responses.

The services listed in this section are built in conformance to the [DICOMweb \(WADO-RS\)](#) standard for web-based medical imaging. Because they are representations of DICOMweb services, they are not offered through AWS CLI and AWS SDKs.

The following table describes all HealthImaging representations of DICOMweb WADO-RS services available for retrieving data from HealthImaging.

HealthImaging representations of DICOMweb WADO-RS services

Name	Description
GetDICOMInstance	Retrieve a DICOM instance (.dcm file) from a HealthImaging data store by specifying the Series, Study, and Instance UIDs associated with a resource.
GetDICOMInstanceMetadata	Retrieve DICOM instance metadata (.json file) from a DICOM instance in a HealthImaging data store by specifying the Series, Study, and Instance UIDs associated with a resource.
GetDICOMInstanceFrames	Retrieve single or batch image frames (multipart request) from a DICOM instance in a HealthImaging data store by specifying the Series UID, Study UID, Instance UIDs, and frame numbers associated with a resource.

Topics

- [Getting a DICOM instance from HealthImaging](#)
- [Getting DICOM instance metadata from HealthImaging](#)
- [Getting DICOM instance frames from HealthImaging](#)

Getting a DICOM instance from HealthImaging

Use the `GetDICOMInstance` action to retrieve a DICOM instance (.dcm file) from a HealthImaging [data store](#) by specifying the Series, Study, and Instance UIDs associated with the resource. You can

specify the [image set](#) from which an instance resource should be retrieved by providing the image set ID as a query parameter. DICOM data can be retrieved in either its stored transfer syntax or as uncompressed (ELE) format.

Note

With `GetDICOMInstance`, you can interoperate with systems that utilize DICOM Part 10 binaries while simultaneously taking advantage of HealthImaging's [cloud native actions](#).

To get a DICOM instance (.dcm)

1. Collect HealthImaging `datastoreId` and `imageSetId` parameter values.
2. Use the [GetImageSetMetadata](#) action with the `datastoreId` and `imageSetId` parameter values to retrieve associated metadata values for `studyInstanceUID`, `seriesInstanceUID`, and `sopInstanceUID`. For more information, see [Getting image set metadata](#).
3. Construct a URL for the request using the values for `datastoreId`, `studyInstanceUID`, `seriesInstanceUID`, `sopInstanceUID`, and `imageSetId`. To view the entire URL path in the following example, scroll over the **Copy** button. The URL is of the form:

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

4. Prepare and send your request. `GetDICOMInstance` uses a HTTP GET request with [AWS Signature Version 4](#) signing protocol. The following code example uses the `curl` command line tool to get a DICOM instance (.dcm file) from 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

The `transfer-syntax` UID is optional and defaults to Explicit VR Little Endian if not included. Supported transfer syntaxes include:

- Explicit VR Little Endian (ELE) - 1.2.840.10008.1.2.1 (default)
- High-Throughput JPEG 2000 with RPCL Options Image Compression (Lossless Only) - 1.2.840.10008.1.2.4.202
- JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8-bit Image Compression - 1.2.840.10008.1.2.4.50 - if the instance is stored in HealthImaging as 1.2.840.10008.1.2.4.50
- JPEG 2000 Image Compression - 1.2.840.10008.1.2.4.91 - if the instance is stored in HealthImaging as 1.2.840.10008.1.2.4.91
- High-Throughput JPEG 2000 Image Compression - 1.2.840.10008.1.2.4.203 - if the instance is stored in HealthImaging as 1.2.840.10008.1.2.4.203

For more information, see [Supported transfer syntaxes](#) and [HTJ2K decoding libraries for AWS HealthImaging](#).

Getting DICOM instance metadata from HealthImaging

Use the `GetDICOMInstanceMetadata` action to retrieve the metadata from a DICOM instance in a HealthImaging [data store](#) by specifying the Series, Study, and Instance UIDs associated with the resource. You can specify the [image set](#) from which instance resource metadata should be retrieved by providing the image set ID as a query parameter.

Note

With `GetDICOMInstanceMetadata`, you can interoperate with systems that utilize DICOM Part 10 binaries while simultaneously taking advantage of HealthImaging's [cloud native actions](#).

To get DICOM instance metadata (.json)

1. Collect HealthImaging `datastoreId` and `imageSetId` parameter values.
2. Use the [GetImageSetMetadata](#) action with the `datastoreId` and `imageSetId` parameter values to retrieve associated metadata values for `studyInstanceUID`, `seriesInstanceUID`, and `sopInstanceUID`. For more information, see [Getting image set metadata](#).
3. Construct a URL for the request using the values for `datastoreId`, `studyInstanceUID`, `seriesInstanceUID`, `sopInstanceUID`, and `imageSetId`. To view the entire URL path in the following example, scroll over the **Copy** button. The URL is of the form:

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

4. Prepare and send your request. `GetDICOMInstanceMetadata` uses a HTTP GET request with [AWS Signature Version 4](#) signing protocol. The following code example uses the `curl` command line tool to get DICOM instance metadata (.json file) from HealthImaging.

Shell

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

Note

The Transfer Syntax UID indicated in the metadata matches the Stored Transfer Syntax UID (StoredTransferSyntaxUID) in HealthImaging.

Getting DICOM instance frames from HealthImaging

Use the `GetDICOMInstanceFrames` action to retrieve single or batch image frames (multipart request) from a DICOM instance in a HealthImaging [data store](#) by specifying the Series UID, Study UID, Instance UIDs, and frame numbers associated with a resource. You can specify the [image set](#) from which instance frames should be retrieved by providing the image set ID as a query parameter. DICOM data can be retrieved in either its stored transfer syntax or as uncompressed (ELE) format.

Note

With `GetDICOMInstanceFrames`, you can interoperate with systems that utilize DICOM Part 10 binaries while simultaneously taking advantage of HealthImaging's [cloud native actions](#).

To get DICOM instance frames (multipart)

1. Collect HealthImaging `datastoreId` and `imageSetId` parameter values.
2. Use the [GetImageSetMetadata](#) action with the `datastoreId` and `imageSetId` parameter values to retrieve associated metadata values for `studyInstanceUID`, `seriesInstanceUID`, and `sopInstanceUID`. For more information, see [Getting image set metadata](#).
3. Determine the image frames to retrieve from the associated metadata to form the `frameList` parameter. The `frameList` parameter is a comma-separated list of one or more non-duplicate frame numbers, in any order. For example, the first image frame in the metadata will be frame 1.
 - Single-frame request: `/frames/1`
 - Multi-frame request: `/frames/1,2,3,4`

- Construct a URL for the request using the values for `datastoreId`, `studyInstanceUID`, `seriesInstanceUID`, `sopInstanceUID`, `imageSetId`, and `frameList`. To view the entire URL path in the following example, scroll over the **Copy** button. The URL is of the form:

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

- Prepare and send your request. `GetDICOMInstanceFrames` uses a HTTP GET request with [AWS Signature Version 4](#) signing protocol. The following code example uses the `curl` command line tool to get image frames in a multipart response from 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

The `transfer-syntax` UID is optional and defaults to Explicit VR Little Endian if not included. Supported transfer syntaxes include:

- Explicit VR Little Endian (ELE) - 1.2.840.10008.1.2.1 (default)
- High-Throughput JPEG 2000 with RPCL Options Image Compression (Lossless Only) - 1.2.840.10008.1.2.4.202
- JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8-bit Image Compression - 1.2.840.10008.1.2.4.50 - if the instance is stored in HealthImaging as 1.2.840.10008.1.2.4.50

- JPEG 2000 Image Compression - 1.2.840.10008.1.2.4.91 - if the instance is stored in HealthImaging as 1.2.840.10008.1.2.4.91
- High-Throughput JPEG 2000 Image Compression - 1.2.840.10008.1.2.4.203 - if the instance is stored in HealthImaging as 1.2.840.10008.1.2.4.203

For more information, see [Supported transfer syntaxes](#) and [HTJ2K decoding libraries for AWS HealthImaging](#).

Monitoring AWS HealthImaging

Monitoring and logging are important parts of maintaining the security, reliability, availability, and performance of AWS HealthImaging. AWS provides the following logging and monitoring tools to watch HealthImaging, report when something is wrong, and take automatic actions when appropriate:

- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).
- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see the [Amazon EventBridge User Guide](#).

Topics

- [Using AWS CloudTrail with HealthImaging](#)
- [Using Amazon CloudWatch with HealthImaging](#)
- [Using Amazon EventBridge with HealthImaging](#)

Using AWS CloudTrail with HealthImaging

AWS HealthImaging is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in HealthImaging. CloudTrail captures all API calls for HealthImaging as events. The calls captured include calls from the HealthImaging console and

code calls to the HealthImaging API operations. If you create a trail, you can turn on continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for HealthImaging. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to HealthImaging, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Creating a trail

CloudTrail is turned on for your AWS account when you create the account. When activity occurs in HealthImaging, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail Event history](#).

Note

To view CloudTrail event history for AWS HealthImaging in the AWS Management Console, go to the **Lookup attributes** menu, select **Event source**, and choose `medical-imaging.amazonaws.com`.

For an ongoing record of events in your AWS account, including events for HealthImaging, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple regions](#) and [Receiving CloudTrail log files from multiple accounts](#)

Note

AWS HealthImaging supports two types of CloudTrail events — management events and data events. *Management events* are the general events that every AWS service generates, including HealthImaging. By default, logging is applied to management events for every HealthImaging API call that has it enabled. *Data events* are billable and generally reserved for APIs that have high transactions per second (tps), so you can opt out of having CloudTrail logs for cost purposes.

With HealthImaging, all API actions listed in the [AWS HealthImaging API Reference](#) are considered management events with the exception of [GetImageFrame](#). The `GetImageFrame` action is onboarded with CloudTrail as a data event and therefore must be enabled. For more information, see [Logging data events](#) in the *AWS CloudTrail User Guide*.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail `userIdentity` element](#).

Understanding log entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry for HealthImaging that demonstrates the `GetDICOMImportJob` action.

```
{  
  "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"
}
```

Using Amazon CloudWatch with HealthImaging

You can monitor AWS HealthImaging using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so you can use that historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

Note

Metrics are reported for all HealthImaging APIs.

The following tables list the metrics and dimensions for HealthImaging. Each is presented as a frequency count for a user-specified data range.

Metrics

Metrics	Description
Call Count	<p>The number of calls to APIs. This can be reported either for the account or a specified data store.</p> <p>Units: Count</p> <p>Valid Statistics: Sum, Count</p> <p>Dimensions: Operation, data store ID, data store type</p>

You can get metrics for HealthImaging with the AWS Management Console, the AWS CLI, or the CloudWatch API. You can use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools. The HealthImaging console displays graphs based on the raw data from the CloudWatch API.

You must have the appropriate CloudWatch permissions to monitor HealthImaging with CloudWatch. For more information, see [Identity and access management for CloudWatch](#) in the *CloudWatch User Guide*.

Viewing HealthImaging metrics

To view metrics (CloudWatch console)

1. Sign in to the AWS Management Console and open the [CloudWatch console](#).
2. Choose **Metrics**, choose **All Metrics**, and then choose **AWS/Medical Imaging**.
3. Choose the dimension, choose a metric name, then choose **Add to graph**.
4. Choose a value for the date range. The metric count for the selected date range is displayed in the graph.

Creating an alarm using CloudWatch

A CloudWatch alarm watches a single metric over a specified time period, and performs one or more actions: sending a notification to an Amazon Simple Notification Service (Amazon SNS) topic or Auto Scaling policy. The action or actions are based on the value of the metric relative to a given threshold over a number of time periods that you specify. CloudWatch can also send you an Amazon SNS message when the alarm changes state.

CloudWatch alarms invoke actions only when the state changes and has persisted for the period you specify. For more information, see [Using CloudWatch alarms](#).

Using Amazon EventBridge with HealthImaging

Amazon EventBridge is a serverless service that uses events to connect application components together, making it easier for you to build scalable event-driven applications. The basis of EventBridge is to create [rules](#) that route [events](#) to [targets](#). AWS HealthImaging provides durable delivery of state changes to EventBridge. For more information, see [What is Amazon EventBridge?](#) in the *Amazon EventBridge User Guide*.

Topics

- [HealthImaging events sent to EventBridge](#)
- [HealthImaging event structure and examples](#)

HealthImaging events sent to EventBridge

The following table lists all HealthImaging events sent to EventBridge for processing.

HealthImaging event type	State
Data store events	
Data Store Creating	CREATING
Data Store Creation Failed	CREATE_FAILED
Data Store Created	ACTIVE
Data Store Deleting	DELETING
Data Store Deleted	DELETED

For more information, see [datastoreStatus](#) in the *AWS HealthImaging API Reference*.

Import job events	
Import Job Submitted	SUBMITTED
Import Job In Progress	IN_PROGRESS
Import Job Completed	COMPLETED
Import Job Failed	FAILED

For more information, see [jobStatus](#) in the *AWS HealthImaging API Reference*.

Image set events	
Image Set Created	CREATED
Image Set Copying	COPYING
Image Set Copying With Read Only Access	COPYING_WITH_READ_ONLY_ACCESS
Image Set Copied	COPIED

HealthImaging event type	State
Image Set Copy Failed	COPY_FAILED
Image Set Updating	UPDATING
Image Set Updated	UPDATED
Image Set Update Failed	UPDATE_FAILED
Image Set Deleting	DELETING
Image Set Deleted	DELETED

For more information, see [ImageSetWorkflowStatus](#) in the *AWS HealthImaging API Reference*.

HealthImaging event structure and examples

HealthImaging events are objects with JSON structure that also contain metadata details. You can use the metadata as input to either recreate an event or learn more information. All associated metadata fields are listed in a table under the code examples in the following menus. For more information, see [Event structure reference](#) in the *Amazon EventBridge User Guide*.

Note

The source attribute for HealthImaging event structures is `aws.medical-imaging`.

Data store events

Data Store Creating

State - 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

State - 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

State - 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

State - 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

State - DELETED

```

{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Data Store Deleted",

```

```

"source": "aws.medical-imaging",
"account": "111122223333",
"time": "2024-03-14T00:01:00Z",
"region": "us-west-2",
"resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
"detail": {
  "imagingVersion": "1.0",
  "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
  "datastoreName": "test",
  "datastoreStatus": "DELETED"
}
}

```

Data store events - metadata descriptions

Name	Type	Description
version	string	The EventBridge event schema version.
id	string	The Version 4 UUID generated for every event.
detail-type	string	The type of event that is being sent.
source	string	Identifies the service that generated the event.
account	string	The 12-digit AWS account ID of the data store owner.
time	string	The time the event occurred.
region	string	Identifies the AWS Region of the data store.
resources	array (string)	A JSON array that contains the ARN of the data store.

Name	Type	Description
detail	object	A JSON object that contains information about the event.
detail.imagingVersion	string	The version ID that tracks changes to HealthImaging's event detail schema.
detail.datastoreId	string	The data store ID associated with the status change event.
detail.datastoreName	string	The data store name.
detail.datastoreStatus	string	The current data store status.

Import job events

Import Job Submitted

State - 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

State - 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

State - 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

State - 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/"
  }
}

```

Import job events - metadata descriptions

Name	Type	Description
version	string	The EventBridge event schema version.

Name	Type	Description
id	string	The Version 4 UUID generated for every event.
detail-type	string	The type of event that is being sent.
source	string	Identifies the service that generated the event.
account	string	The 12-digit AWS account ID of the data store owner.
time	string	The time the event occurred.
region	string	Identifies the AWS Region of the data store.
resources	array (string)	A JSON array that contains the ARN of the data store.
detail	object	A JSON object that contains information about the event.
detail.imagingVersion	string	The version ID that tracks changes to HealthImaging's event detail schema.
detail.datastoreId	string	The data store that generated the status change event.
detail.jobId	string	The import job ID associated with the status change event.
detail.jobName	string	The import job name.
detail.jobStatus	string	The current job status.

Name	Type	Description
detail.inputS3Uri	string	The input prefix path for the S3 bucket that contains the DICOM files to be imported.
detail.outputS3Uri	string	The output prefix of the S3 bucket where the results of the DICOM import job will be uploaded.

Image set events

Image Set Created

State - CREATED

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Created",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "CREATED"
  }
}
```

Image Set Copying

State - COPYING

```
{
```

```

"version": "0",
"id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
"detail-type": "Image Set Copying",
"source": "aws.medical-imaging",
"account": "111122223333",
"time": "2024-03-14T00:01:00Z",
"region": "us-west-2",
"resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
"detail": {
  "imagingVersion": "1.0",
  "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
  "imagesetId": "5b3a711878c34d40e888253319388649",
  "imageSetState": "LOCKED",
  "imageSetWorkflowStatus": "COPYING"
}
}

```

Image Set Copying With Read Only Access

State - COPYING_WITH_READ_ONLY_ACCESS

```

{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Copying With Read Only Access",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS"
  }
}

```

Image Set Copied

State - COPIED

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Copied",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "COPIED"
  }
}
```

Image Set Copy Failed

State - COPY_FAILED

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Copy Failed",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "COPY_FAILED"
  }
}
```

Image Set Updating

State - UPDATING

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Updating",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "UPDATING"
  }
}
```

Image Set Updated

State - UPDATED

```
{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Updated",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "UPDATED"
  }
}
```

```
}  
}
```

Image Set Update Failed

State - UPDATE_FAILED

```
{  
  "version": "0",  
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",  
  "detail-type": "Image Set Update Failed",  
  "source": "aws.medical-imaging",  
  "account": "111122223333",  
  "time": "2024-03-14T00:01:00Z",  
  "region": "us-west-2",  
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/  
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],  
  "detail": {  
    "imagingVersion": "1.0",  
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",  
    "imagesetId": "5b3a711878c34d40e888253319388649",  
    "imageSetState": "ACTIVE",  
    "imageSetWorkflowStatus": "UPDATE_FAILED"  
  }  
}
```

Image Set Deleting

State - DELETING

```
{  
  "version": "0",  
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",  
  "detail-type": "Image Set Deleting",  
  "source": "aws.medical-imaging",  
  "account": "111122223333",  
  "time": "2024-03-14T00:01:00Z",  
  "region": "us-west-2",  
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/  
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],  
  "detail": {  
    "imagingVersion": "1.0",  
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
```

```

    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "LOCKED",
    "imageSetWorkflowStatus": "DELETING"
  }
}

```

Image Set Deleted

State - DELETED

```

{
  "version": "0",
  "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
  "detail-type": "Image Set Deleted",
  "source": "aws.medical-imaging",
  "account": "111122223333",
  "time": "2024-03-14T00:01:00Z",
  "region": "us-west-2",
  "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "DELETED",
    "imageSetWorkflowStatus": "DELETED"
  }
}

```

Image set events - metadata descriptions

Name	Type	Description
version	string	The EventBridge event schema version.
id	string	The Version 4 UUID generated for every event.
detail-type	string	The type of event that is being sent.

Name	Type	Description
source	string	Identifies the service that generated the event.
account	string	The 12-digit AWS account ID of the data store owner.
time	string	The time the event occurred.
region	string	Identifies the AWS Region of the data store.
resources	array (string)	A JSON array that contains the ARN of the image set.
detail	object	A JSON object that contains information about the event.
detail.imagingVersion	string	The version ID that tracks changes to HealthImaging's event detail schema.
detail.datastoreId	string	The data store ID that generated the status change event.
detail.imagesetId	string	The image set ID associated with the status change event.
detail.imageSetState	string	The current image set state.
detail.imageSetWorkflowStatus	string	The current image set workflow status.

Security in AWS HealthImaging

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS HealthImaging, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using HealthImaging. The following topics show you how to configure HealthImaging to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your HealthImaging resources.

Topics

- [Data protection in AWS HealthImaging](#)
- [Identity and Access Management for AWS HealthImaging](#)
- [Compliance validation for AWS HealthImaging](#)
- [Infrastructure security in AWS HealthImaging](#)
- [Creating AWS HealthImaging resources with AWS CloudFormation](#)
- [AWS HealthImaging and interface VPC endpoints \(AWS PrivateLink\)](#)
- [Cross-account import for AWS HealthImaging](#)
- [Resilience in AWS HealthImaging](#)

Data protection in AWS HealthImaging

The AWS [shared responsibility model](#) applies to data protection in AWS HealthImaging. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with HealthImaging or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Topics

- [Data encryption](#)
- [Network traffic privacy](#)

Data encryption

With AWS HealthImaging, you can add a layer of security to your data at rest in the cloud, providing scalable and efficient encryption features. These include:

- Data at rest encryption capabilities available in most AWS services
- Flexible key management options, including AWS Key Management Service, with which you can choose whether to have AWS manage the encryption keys or to keep complete control over your own keys.
- AWS owned AWS KMS encryption keys
- Encrypted message queues for the transmission of sensitive data using server-side encryption (SSE) for Amazon SQS

In addition, AWS provides APIs for you to integrate encryption and data protection with any of the services you develop or deploy in an AWS environment.

Encryption at rest

By default, HealthImaging encrypts customer data at rest using a service-owned AWS Key Management Service key. Optionally, you can configure HealthImaging to encrypt data at rest using a symmetric customer-managed AWS KMS key that you create, own, and manage. For more information, see [Create a symmetric encryption KMS key](#) in the *AWS Key Management Service Developer Guide*.

Encryption in transit

HealthImaging uses TLS 1.2 to encrypt data in transit through the public endpoint and through backend services.

Key management

AWS KMS keys (KMS keys) are the primary resource in AWS Key Management Service. You can also generate data keys for use outside of AWS KMS.

AWS owned KMS key

HealthImaging uses these keys by default to automatically encrypt potentially sensitive information such as personally identifiable or Private Health Information (PHI) data at rest. AWS owned KMS keys aren't stored in your account. They're part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned KMS keys to protect your data. You can't view, manage, use AWS owned KMS keys, or audit their use. However, you don't need to do any work or change any programs to protect the keys that encrypt your data.

You're not charged a monthly fee or a usage fee if you use AWS owned KMS keys, and they don't count against AWS KMS quotas for your account. For more information, see [AWS owned keys](#) in the *AWS Key Management Service Developer Guide*.

Customer managed KMS keys

If you want full control over AWS KMS lifecycle and usage, HealthImaging supports the use of a symmetric customer-managed KMS key that you create, own, and manage. Because you have full control of this layer of encryption, you can perform such tasks as:

- Establishing and maintaining key policies, IAM policies, and grants
- Rotating key cryptographic material
- Enabling and disabling key policies
- Adding tags
- Creating key aliases
- Scheduling keys for deletion

You can also use CloudTrail to track the requests that HealthImaging sends to AWS KMS on your behalf. Additional AWS KMS charges apply. For more information, see [Customer managed keys](#) in the *AWS Key Management Service Developer Guide*.

Creating a customer managed key

You can create a symmetric customer managed key by using the AWS Management Console or the AWS KMS APIs. For more information, see [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*.

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how

they can use it. When you create your customer managed key, you can specify a key policy. For more information, see [Managing access to customer managed keys](#) in the *AWS Key Management Service Developer Guide*.

To use your customer managed key with your HealthImaging resources, [kms:CreateGrant](#) operations must be permitted in the key policy. This adds a grant to a customer managed key which controls access to a specified KMS key, which gives a user access to the [Grant operations](#) HealthImaging requires. For more information, see [Grants in AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

To use your customer managed KMS key with your HealthImaging resources, the following API operations must be permitted in the key policy:

- `kms:DescribeKey` provides the customer managed key details needed to validate the key. This is required for all operations.
- `kms:GenerateDataKey` provides access to encrypt resources at rest for all write operations.
- `kms:Decrypt` provides access to read or search operations for encrypted resources.
- `kms:ReEncrypt*` provides access to reencrypt resources.

The following is a policy statement example that allows a user to create and interact with a data store in HealthImaging which is encrypted by that key:

```
{
  "Sid": "Allow access to create data stores and perform CRUD and search in
HealthImaging",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "medical-imaging.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:kms-arn": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f",
```

```
        "kms:EncryptionContext:aws:medical-imaging:datastoreId": "datastoreId"
      }
    }
  }
```

Required IAM permissions for using a customer managed KMS key

When creating a data store with AWS KMS encryption enabled using a customer managed KMS key, there are required permissions for both the key policy and the IAM policy for the user or role creating the HealthImaging data store.

For more information about key policies, see [Enabling IAM policies](#) in the *AWS Key Management Service Developer Guide*.

The IAM user, IAM role, or AWS account creating your repositories must have permissions for `kms:CreateGrant`, `kms:GenerateDataKey`, `kms:RetireGrant`, `kms:Decrypt`, and `kms:ReEncrypt*`, plus the necessary permissions for AWS HealthImaging.

How HealthImaging uses grants in AWS KMS

HealthImaging requires a [grant](#) to use your customer managed KMS key. When you create a data store encrypted with a customer managed KMS key, HealthImaging creates a grant on your behalf by sending a [CreateGrant](#) request to AWS KMS. Grants in AWS KMS are used to give HealthImaging access to a KMS key in a customer account.

The grants that HealthImaging creates on your behalf should not be revoked or retired. If you revoke or retire the grant that gives HealthImaging permission to use the AWS KMS keys in your account, HealthImaging cannot access this data, encrypt new imaging resources pushed to the data store, or decrypt them when they are pulled. When you revoke or retire a grant for HealthImaging, the change occurs immediately. To revoke access rights, you should delete the data store rather than revoke the grant. When a data store is deleted, HealthImaging retires the grants on your behalf.

Monitoring your encryption keys for HealthImaging

You can use CloudTrail to track the requests that HealthImaging sends to AWS KMS on your behalf when using a customer managed KMS key. The log entries in the CloudTrail log show `medical-imaging.amazonaws.com` in the `userAgent` field to clearly distinguish requests made by HealthImaging.

The following examples are CloudTrail events for `CreateGrant`, `GenerateDataKey`, `Decrypt`, and `DescribeKey` to monitor AWS KMS operations called by HealthImaging to access data encrypted by your customer managed key.

The following shows how to use `CreateGrant` to allow HealthImaging to access a customer provided KMS key, enabling HealthImaging to use that KMS key to encrypt all customer data at rest.

Users are not required to create their own grants. HealthImaging creates a grant on your behalf by sending a `CreateGrant` request to AWS KMS. Grants in AWS KMS are used to give HealthImaging access to a AWS KMS key in a customer account.

```
{
  "Grants": [
    {
      "Operations": [
        "Decrypt",
        "Encrypt",
        "GenerateDataKey",
        "GenerateDataKeyWithoutPlaintext",
        "DescribeKey"
      ],
      "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-b5841e1181d1",
      "Name": "0a74e6ad2aa84b74a22fcd3efac1eaa8",
      "RetiringPrincipal": "AWS Internal",
      "GranteePrincipal": "AWS Internal",
      "GrantId":
"0da169eb18ffd3da8c0eebc9e74b3839573eb87e1e0dce893bb544a34e8fbaaf",
      "IssuingAccount": "AWS Internal",
      "CreationDate": 1685050229.0,
      "Constraints": {
        "EncryptionContextSubset": {
          "kms-arn": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-b5841e1181d1"
        }
      }
    },
    {
      "Operations": [
        "GenerateDataKey",
        "CreateGrant",
        "RetireGrant",

```

```

        "DescribeKey"
      ],
      "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
      "Name": "2023-05-25T21:30:17",
      "RetiringPrincipal": "AWS Internal",
      "GranteePrincipal": "AWS Internal",
      "GrantId":
"8229757abbb2019555ba64d200278cedac08e5a7147426536fcd1f4270040a31",
      "IssuingAccount": "AWS Internal",
      "CreationDate": 1685050217.0,
    }
  ]
}

```

The following examples shows how to use `GenerateDataKey` to ensure the user has necessary permissions to encrypt data before storing it.

```

{
  "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"
}

```

The following example shows how HealthImaging calls the Decrypt operation to use the stored encrypted data key to access the encrypted data.

```

{
  "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"
}

```

The following example shows how HealthImaging uses the `DescribeKey` operation to verify if the AWS KMS customer owned AWS KMS key is in a usable state and to help a user troubleshoot if it is not functional.

```

{
    "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"  
}
```

Learn more

The following resources provide more information about data at rest encryption and are located in the *AWS Key Management Service Developer Guide*.

- [AWS KMS concepts](#)
- [Security best practices for AWS KMS](#)

Network traffic privacy

Traffic is protected both between HealthImaging and on-premises applications and between HealthImaging and Amazon S3. Traffic between HealthImaging and AWS Key Management Service uses HTTPS by default.

- **AWS HealthImaging is a regional service** available in the US East (N. Virginia), US West (Oregon), Europe (Ireland), and Asia Pacific (Sydney) Regions.
- **For traffic between HealthImaging and Amazon S3 buckets**, Transport Layer Security (TLS) encrypts objects in-transit between HealthImaging and Amazon S3, and between HealthImaging and customer applications accessing it, you should allow only encrypted connections over HTTPS (TLS) using the [aws:SecureTransport condition](#) on Amazon S3 bucket IAM policies. Although HealthImaging currently uses the public endpoint to access data in Amazon S3 buckets, this does not mean that the data traverses the public internet. All traffic between HealthImaging and Amazon S3 is routed over the AWS network and is encrypted using TLS.

Identity and Access Management for AWS HealthImaging

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use HealthImaging resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How AWS HealthImaging works with IAM](#)
- [Identity-based policy examples for AWS HealthImaging](#)
- [AWS managed policies for AWS HealthImaging](#)
- [Troubleshooting AWS HealthImaging identity and access](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in HealthImaging.

Service user – If you use the HealthImaging service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more HealthImaging features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in HealthImaging, see [Troubleshooting AWS HealthImaging identity and access](#).

Service administrator – If you're in charge of HealthImaging resources at your company, you probably have full access to HealthImaging. It's your job to determine which HealthImaging features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with HealthImaging, see [How AWS HealthImaging works with IAM](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to HealthImaging. To view example HealthImaging identity-based policies that you can use in IAM, see [Identity-based policy examples for AWS HealthImaging](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on

authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [AWS Multi-factor authentication in IAM](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [Use cases for IAM users](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can [switch from a user to an IAM role \(console\)](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#)

in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
 - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
 - **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
 - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile

that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choose between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are *IAM role trust policies* and *Amazon S3 bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to

any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [Service control policies](#) in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How AWS HealthImaging works with IAM

Before you use IAM to manage access to HealthImaging, learn what IAM features are available to use with HealthImaging.

IAM features you can use with AWS HealthImaging

IAM feature	HealthImaging support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes

IAM feature	HealthImaging support
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how HealthImaging and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for HealthImaging

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for HealthImaging

To view examples of HealthImaging identity-based policies, see [Identity-based policy examples for AWS HealthImaging](#).

Resource-based policies within HealthImaging

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#)

in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for HealthImaging

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of HealthImaging actions, see [Actions defined by AWS HealthImaging](#) in the *Service Authorization Reference*.

Policy actions in HealthImaging use the following prefix before the action:

```
AWS
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
  "AWS:action1",  
  "AWS:action2"  
]
```

To view examples of HealthImaging identity-based policies, see [Identity-based policy examples for AWS HealthImaging](#).

Policy resources for HealthImaging

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*" 
```

To see a list of HealthImaging resource types and their ARNs, see [Resource types defined by AWS HealthImaging](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use an ARN, see [Actions defined by AWS HealthImaging](#).

To view examples of HealthImaging identity-based policies, see [Identity-based policy examples for AWS HealthImaging](#).

Policy condition keys for HealthImaging

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition *block*) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of HealthImaging condition keys, see [Condition keys for AWS HealthImaging](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by AWS HealthImaging](#).

To view examples of HealthImaging identity-based policies, see [Identity-based policy examples for AWS HealthImaging](#).

ACLs in HealthImaging

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

RBAC with HealthImaging

Supports RBAC

Yes

The traditional authorization model used in IAM is called role-based access control (RBAC). RBAC defines permissions based on a person's job function, known outside of AWS as a *role*. For more information, see [Comparing ABAC to the traditional RBAC model](#) in the *IAM User Guide*.

ABAC with HealthImaging

Supports ABAC (tags in policies): Partial

Warning

ABAC is not enforced via the SearchImageSets API action. Anyone who has access to the SearchImageSets action can access all metadata for image sets in a data store.

Note

Image sets are a child resource of data stores. To use ABAC, an image set must have the same tag as a data store. For more information, refer to [Tagging resources with AWS HealthImaging](#).

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [Define permissions with ABAC authorization](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

Using temporary credentials with HealthImaging

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switch from a user to an IAM role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for HealthImaging

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for AWS HealthImaging](#) in the *Service Authorization Reference*.

Service roles for HealthImaging

Supports service roles: Yes

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break HealthImaging functionality. Edit service roles only when HealthImaging provides guidance to do so.

Service-linked roles for HealthImaging

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](#). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS HealthImaging

By default, users and roles don't have permission to create or modify HealthImaging resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Create IAM policies \(console\)](#) in the *IAM User Guide*.

For details about actions and resource types defined by Awesome, including the format of the ARNs for each of the resource types, see [Actions, Resources, and Condition Keys for AWS Awesome](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)
- [Using the HealthImaging console](#)
- [Allow users to view their own permissions](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete HealthImaging resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the HealthImaging console

To access the AWS HealthImaging console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the HealthImaging resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the HealthImaging console, also attach the HealthImaging *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
    },
  ],
}
```

```
        "Resource": "*"
    }
  ]
}
```

AWS managed policies for AWS HealthImaging

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see [AWS managed policies](#) in the *IAM User Guide*.

Topics

- [AWS managed policy: AWSHealthImagingFullAccess](#)
- [AWS managed policy: AWSHealthImagingReadOnlyAccess](#)
- [HealthImaging updates to AWS managed policies](#)

AWS managed policy: AWSHealthImagingFullAccess

You can attach the `AWSHealthImagingFullAccess` policy to your IAM identities.

This policy grants administrative permission to all HealthImaging actions.

```
{
  "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 managed policy: `AWSHealthImagingReadOnlyAccess`

You can attach the `AWSHealthImagingReadOnlyAccess` policy to your IAM identities.

This policy grants read-only permission to specific AWS HealthImaging actions.

```
{
  "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 updates to AWS managed policies

View details about updates to AWS managed policies for HealthImaging since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [Releases](#) page.

Change	Description	Date
HealthImaging started tracking changes	HealthImaging started tracking changes for its AWS managed policies.	July 19, 2023

Troubleshooting AWS HealthImaging identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with HealthImaging and IAM.

Topics

- [I am not authorized to perform an action in HealthImaging](#)
- [I am not authorized to perform iam:PassRole](#)

- [I want to allow people outside of my AWS account to access my HealthImaging resources](#)

I am not authorized to perform an action in HealthImaging

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional `my-example-widget` resource but doesn't have the fictional AWS: `GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
AWS: GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the `my-example-widget` resource by using the AWS: `GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to HealthImaging.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in HealthImaging. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my HealthImaging resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether HealthImaging supports these features, see [How AWS HealthImaging works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Compliance validation for AWS HealthImaging

Third-party auditors assess the security and compliance of AWS HealthImaging as part of multiple AWS compliance programs. For HealthImaging, this includes HIPAA.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS HealthImaging is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [AWS Partner Solutions](#) – The automated reference deployment guides for Security and Compliance discuss architectural considerations and provide steps for deploying security and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [GxP Systems on AWS](#) – This whitepaper provides information on how AWS approaches GxP-related compliance and security and provides guidance on using AWS services in the context of GxP.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) – AWS Config assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Infrastructure security in AWS HealthImaging

As a managed service, AWS HealthImaging is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access HealthImaging through the network. Clients must support Transport Layer Security (TLS) 1.3 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed using an access key ID and a secret access key that is associated with an IAM principal. You can also use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Creating AWS HealthImaging resources with AWS CloudFormation

AWS HealthImaging is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources

and infrastructure. You create a template that describes all the AWS resources that you want and AWS CloudFormation provisions and configures those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your HealthImaging resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

HealthImaging and AWS CloudFormation templates

To provision and configure resources for HealthImaging and related services, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the *AWS CloudFormation User Guide*.

AWS HealthImaging supports creating [data stores](#) with AWS CloudFormation. For more information, including examples of JSON and YAML templates for provisioning HealthImaging data stores, see the [AWS HealthImaging resource type reference](#) in the *AWS CloudFormation User Guide*.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation API Reference](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

AWS HealthImaging and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and AWS HealthImaging by creating an *interface VPC endpoint*. Interface endpoints are powered by [AWS PrivateLink](#), a technology that you can use to privately access HealthImaging APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP

addresses to communicate with HealthImaging APIs. Traffic between your VPC and HealthImaging does not leave the Amazon network.

Each interface endpoint is represented by one or more [Elastic Network Interfaces](#) in your subnets.

For more information, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.

Topics

- [Considerations for HealthImaging VPC endpoints](#)
- [Creating an interface VPC endpoint for HealthImaging](#)
- [Creating a VPC endpoint policy for HealthImaging](#)

Considerations for HealthImaging VPC endpoints

Before you set up an interface VPC endpoint for HealthImaging, make sure that you review [Interface endpoint properties and limitations](#) in the *Amazon VPC User Guide*.

HealthImaging supports making calls to all AWS HealthImaging actions from your VPC.

Creating an interface VPC endpoint for HealthImaging

You can create a VPC endpoint for the HealthImaging service using the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*.

Create VPC endpoints for HealthImaging using the following service names:

- `com.amazonaws.region.medical-imaging`
- `com.amazonaws.region.runtime-medical-imaging`
- `com.amazonaws.region.dicom-medical-imaging`

Note

Private DNS must be enabled to use PrivateLink.

You can make API requests to HealthImaging using its default DNS name for the Region, for example, `medical-imaging.us-east-1.amazonaws.com`.

For more information, see [Accessing a service through an interface endpoint](#) in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for HealthImaging

You can attach an endpoint policy to your VPC endpoint that controls access to HealthImaging. The policy specifies the following information:

- The principal that can perform actions
- The actions that can be performed
- The resources on which actions can be performed

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for HealthImaging actions

The following is an example of an endpoint policy for HealthImaging. When attached to an endpoint, this policy grants access to HealthImaging actions for all principals on all resources.

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\": \"*\"}]}"
```

Cross-account import for AWS HealthImaging

With cross-account/cross-region import, you can import data into your HealthImaging [data store](#) from Amazon S3 buckets located in other [supported Regions](#). You can import data across AWS accounts, accounts owned by other [AWS Organizations](#), and from open data sources like [Imaging Data Commons \(IDC\)](#) located in the [Registry of Open Data on AWS](#).

HealthImaging cross-account/cross-region import use cases include:

- Medical imaging SaaS products importing DICOM data from customer accounts
- Large organizations populating one HealthImaging data store from many Amazon S3 input buckets
- Researchers securely sharing data across multi-institution clinical studies

To use cross-account import

1. The Amazon S3 input (source) bucket owner must grant the HealthImaging data store owner `s3:ListBucket` and `s3:GetObject` permissions.
2. The HealthImaging data store owner must add the Amazon S3 bucket to their IAM `ImportJobDataAccessRole`. See [Create an IAM role for import](#).
3. The HealthImaging data store owner must provide the [inputOwnerAccountId](#) for the Amazon S3 input bucket when starting the import job.

Note

By providing the `inputOwnerAccountId`, the data store owner validates the input Amazon S3 bucket belongs to the specified account to maintain compliance with industry standards and mitigate potential security risks.

The following startDICOMImportJob code example includes the optional inputOwnerId parameter, which can be applied to all AWS CLI and SDK code examples in the [Starting an import job](#) section.

Java

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

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

    return "";
}
```

Resilience in AWS HealthImaging

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

In addition to the AWS global infrastructure, AWS HealthImaging offers several features to help support your data resiliency and backup needs.

AWS HealthImaging reference material

The following reference material is available for AWS HealthImaging.

Note

All native HealthImaging actions and data types are described in a separate reference. For more information, see the [AWS HealthImaging API Reference](#).

Topics

- [DICOM support for AWS HealthImaging](#)
- [AWS HealthImaging pixel data verification](#)
- [HTJ2K decoding libraries for AWS HealthImaging](#)
- [AWS HealthImaging endpoints and quotas](#)
- [AWS HealthImaging throttling limits](#)
- [AWS HealthImaging sample projects](#)
- [Using HealthImaging with an AWS SDK](#)

DICOM support for AWS HealthImaging

AWS HealthImaging supports specific DICOM elements and transfer syntaxes. Familiarize yourself with the supported Patient, Study, and Series level DICOM data elements, as HealthImaging metadata keys are based on them. Before you start an import, verify that your medical imaging data is compliant with HealthImaging's supported transfer syntaxes and DICOM element constraints.

Note

AWS HealthImaging does not currently support Binary Segmentation Images or Icon Image Sequence pixel data.

Topics

- [Supported SOP classes](#)

- [Metadata normalization](#)
- [Supported transfer syntaxes](#)
- [DICOM element constraints](#)
- [DICOM metadata constraints](#)

Supported SOP classes

With AWS HealthImaging, you can import DICOM P10 Service-Object Pair (SOP) instances encoded with any SOP class UID, including [retired](#) and [private](#). All private *attributes* are preserved, as well.

Metadata normalization

When you import your DICOM P10 data into AWS HealthImaging, it is transformed into [image sets](#) comprised of [metadata](#) and [image frames](#) (pixel data). During the transformation process, HealthImaging metadata keys are generated based on a specific version of the DICOM standard. HealthImaging currently generates and supports metadata keys based on the [DICOM PS3.6 2022b Data Dictionary](#).

AWS HealthImaging supports the following DICOM data elements at the Patient, Study, and Series levels.

Patient level elements

Note

For a detailed description of each Patient level element, see the [Registry of DICOM Data Elements](#).

AWS HealthImaging supports the following Patient level elements:

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

Study level elements

Note

For a detailed description of each Study level element, see the [Registry of DICOM Data Elements](#).

AWS HealthImaging supports the following Study level elements:

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

Series level elements

Note

For a detailed description of each Series level element, see the [Registry of DICOM Data Elements](#).

AWS HealthImaging supports the following Series level elements:

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
```

Supported transfer syntaxes

AWS HealthImaging supports importing DICOM P10 files with different transfer syntaxes. Some files retain their original transfer syntax encoding during import, while others are transcoded to HTJ2K lossless by default. The following example shows how HealthImaging records a `StoredTransferSyntaxUID` for each instance within the [metadata](#) returned by [GetImageSetMetadata](#).

```
"Instances": {
  "999.999.2.19941105.134500.2.101": {
    "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.50",
    "ImageFrames": [{ ...
```

Note

Keep these points in mind when viewing the following table:

- A transfer syntax UID entry marked with an asterisk (*) indicates a file is stored in its original encoded format during import. For these files, the `StoredTransferSyntaxUID` located in the instance metadata matches the original transfer syntax.
- A transfer syntax UID entry marked *without* an asterisk indicates a file is transcoded to HTJ2K lossless during import and stored in HealthImaging. For these files, the `StoredTransferSyntaxUID` located in the instance metadata is High-Throughput JPEG 2000 with RPCL Options Image Compression—Lossless Only (1.2.840.10008.1.2.4.202).
- If the `StoredTransferSyntaxUID` key does not exist or is set to `null`, you can assume it is encoded as HTJ2K Lossless RPCL (1.2.840.10008.1.2.4.202).

HealthImaging supported transfer syntaxes

Transfer syntax UID	Transfer syntax name
1.2.840.10008.1.2	Implicit VR Endian: Default Transfer Syntax for DICOM
1.2.840.10008.1.2.1* (binary segmentation retains original encoding, whereas, non-binary segmentation is transcoded to HTJ2K Lossless RPCL)	Explicit VR Little Endian
1.2.840.10008.1.2.1.99	Deflated Explicit VR Little Endian
1.2.840.10008.1.2.2	Explicit VR Big Endian
1.2.840.10008.1.2.4.50*	JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8-bit Image Compression
1.2.840.10008.1.2.4.51	JPEG Baseline (Processes 2 & 4): Default Transfer Syntax for Lossy JPEG 12-bit Image Compression (Process 4 only)
1.2.840.10008.1.2.4.57	JPEG Lossless Non-Hierarchical (Process 14)
1.2.840.10008.1.2.4.70	JPEG Lossless, Nonhierarchical, First-Order Prediction (Processes 14 [Selection Value 1]): Default Transfer Syntax for Lossless JPEG Image Compression
1.2.840.10008.1.2.4.80	JPEG-LS Lossless Image Compression
1.2.840.10008.1.2.4.81	JPEG-LS Lossy (Near-Lossless) Image Compression
1.2.840.10008.1.2.4.90	JPEG 2000 Image Compression (Lossless Only)
1.2.840.10008.1.2.4.91*	JPEG 2000 Image Compression

Transfer syntax UID	Transfer syntax name
1.2.840.10008.1.2.4.201	High-Throughput JPEG 2000 Image Compression (Lossless Only)
1.2.840.10008.1.2.4.202	High-Throughput JPEG 2000 with RPCL Options Image Compression (Lossless Only)
1.2.840.10008.1.2.4.203*	High-Throughput JPEG 2000 Image Compression
1.2.840.10008.1.2.5	RLE Lossless

*Retains original transfer syntax encoding during import

DICOM element constraints

When importing your medical imaging data into AWS HealthImaging, max length constraints are applied to the following DICOM elements. To achieve a successful import, ensure that your data does not exceed the max length constraints.

DICOM element constraints during import

HealthImaging keyword	DICOM keyword	DICOM key	Length limit
DICOMPatientName	PatientName	(0010,0010)	min: 0, max: 256
DICOMPatientId	PatientID	(0010,0020)	min: 0, max: 256
DICOMPatientBirthDate	PatientBirthDate	(0010,0030)	min: 0, max: 18
DICOMPatientSex	PatientSex	(0010,0040)	min: 0, max: 16
DICOMStudyInstanceUID	StudyInstanceUID	(0020,000D)	min: 0, max: 256
DICOMStudyId	StudyID	(0020,0010)	min: 0, max: 16

HealthImaging keyword	DICOM keyword	DICOM key	Length limit
DICOMStudyDescription	StudyDescription	(0008,1030)	min: 0, max: 64
DICOMNumberOfStudyRelatedSeries	NumberOfStudyRelatedSeries	(0020,1206)	min: 0, max: 1,000,000
DICOMNumberOfStudyRelatedInstances	NumberOfStudyRelatedInstances	(0020,1208)	min: 0, max: 10,000
DICOMAccessionNumber	AccessionNumber	(0008,0050)	min: 0, max: 256
DICOMStudyDate	StudyDate	(0008,0020)	min: 0, max: 18
DICOMStudyTime	StudyTime	(0008,0030)	min: 0, max: 28

DICOM metadata constraints

When you use `UpdateImageSetMetadata` to update HealthImaging [metadata](#) attributes, the following DICOM constraints are applied.

- Cannot update or remove private attributes on Patient/Study/Series/Instance level attributes unless the update constraint applies to both `updateableAttributes` and `removableAttributes`
- Cannot update the following AWS HealthImaging generated attributes: `SchemaVersion`, `DatastoreID`, `ImageSetID`, `PixelData`, `Checksum`, `Width`, `Height`, `MinPixelValue`, `MaxPixelValue`, `FrameSizeInBytes`
- Cannot update the following DICOM attributes unless the force flag is set: `Tag.PixelData`, `Tag.StudyInstanceUID`, `Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID`, `Tag.StudyID`
- Cannot update attributes with VR type SQ (nested attributes) unless the force flag is set
- Cannot update multivalued attributes unless the force flag is set

- Cannot update attributes with values that are not compatible with the attribute VR type unless the `force` flag is set
- Cannot update attributes which are not considered valid attributes according to the DICOM standard unless the `force` flag is set
- Cannot update attributes across modules. For example, if a Patient level attribute is given at the Study level in customer payload request, the request can be invalidated.
- Cannot update attributes if the associated attribute module is not present in existing `ImageSetMetadata`. For example, you are not allowed to update attributes for a `seriesInstanceUID` if the Series with `seriesInstanceUID` is not present in existing image set metadata.

AWS HealthImaging pixel data verification

During import, HealthImaging provides built-in pixel data verification by checking the lossless encoding and decoding state of every image. This feature ensures that images decoded using [HTJ2K decoding libraries](#) always match the original DICOM P10 images imported into HealthImaging.

- The image onboarding process begins when an import job captures the original pixel quality state of DICOM P10 images before they are imported. A unique immutable Image Frame Resolution Checksum (IFRC) is generated for each image using the CRC32 algorithm. The IFRC checksum value is presented in the `job-output-manifest.json` metadata document. For more information, see [Understanding import jobs](#)
- After the images are imported into a HealthImaging [data store](#) and transformed into [image sets](#), the HTJ2K-encoded [image frames](#) are immediately decoded and new IFRCs are calculated. HealthImaging then compares the full resolution IFRCs of the original images against the new IFRCs of the imported image frames to verify accuracy.
- A corresponding per-image descriptive error condition is captured in the import job output log (`job-output-manifest.json`) for you to review and verify.

To verify pixel data

1. After importing your medical imaging data, view the per-image set descriptive success (or error condition) captured in the import job output log, `job-output-manifest.json`. For more information, see [Understanding import jobs](#).

2. [Image sets](#) are comprised of [metadata](#) and [image frames](#) (pixel data). Image set metadata contains information about associated image frames. Use the `GetImageSetMetadata` action to get metadata for an image set. For more information, see [Getting image set metadata](#).
3. The `PixelDataChecksumFromBaseToFullResolution` contains the IFRC (checksum) for the full resolution image. For images stored in their original transfer syntax 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50, and 1.2.840.10008.1.2.1 (Binary Segmentation only) the checksum is calculated on the original image. For images that are stored in HTJ2K Lossless with RPCL, the checksum is calculated on the decoded full resolution image. For more information, see [Supported transfer syntaxes](#).

Following is example metadata output for the IFRC that is generated as part of the import job process and recorded to `job-output-manifest.json`.

```
"ImageFrames": [{
  "ID": "67890678906789012345123451234512",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 512,
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
}]
```

For images stored in their original transfer syntax 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50, and 1.2.840.10008.1.2.1 (Binary Segmentation only) the `MinPixelValue` and `MaxPixelValue` will not be available. The `FrameSizeInBytes` indicates the size of the original frame.

```
"PixelDataChecksumFromBaseToFullResolution": [
  {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": null,
"MaxPixelValue": null,
"FrameSizeInBytes": 429
```

For images stored in HTJ2K Lossless with RPCL, the `FrameSizeInBytes` indicates the size of the decoded image frame.

```
"PixelDataChecksumFromBaseToFullResolution": [
```

```
    {"Width": 512, "Height": 512, "Checksum": 1379921327 }  
  ],  
  "MinPixelValue": 11,  
  "MaxPixelValue": 11,  
  "FrameSizeInBytes": 1652
```

4. To verify pixel data, access the [Pixel data verification](#) procedure on GitHub and follow the instructions in the README .md file to independently verify lossless image processing by the various [HTJ2K decoding libraries](#) that are utilized by HealthImaging. After the full image is loaded, you can compute the IFRC for raw input data on your end and compare it with the IFRC value provided in the HealthImaging metadata to verify the pixel data.

HTJ2K decoding libraries for AWS HealthImaging

During [import](#), some transfer syntaxes retain their original encoding, while others are transcoded to High-Throughput JPEG 2000 (HTJ2K) lossless by default. HTJ2K delivers consistently fast image display and universal access to HTJ2K's advanced features. Because some image frames are encoded in HTJ2K during import, they must be decoded prior to viewing in an image viewer. For information about determining transfer syntaxes, see [Supported transfer syntaxes](#).

Note

HTJ2K is defined in [Part 15 of the JPEG2000 standard \(ISO/IEC 15444-15:2019\)](#). HTJ2K retains the advanced features of JPEG2000 such as resolution scalability, precincts, tiling, high bit depth, multiple channels, and color space support.

Topics

- [HTJ2K decoding libraries](#)
- [Image viewers](#)

HTJ2K decoding libraries

Depending on your programming language, we recommend the following decoding libraries to decode [image frames](#).

- [NVIDIA nvJPEG2000](#) – Commercial, GPU-accelerated

- [Kakadu Software](#) – Commercial, C++ with Java and .NET bindings
- [OpenJPH](#) – Open source, C++ and WASM
- [OpenJPEG](#) – Open source, C/C++, Java
- [openjphpy](#) – Open source, Python
- [pylibjpeg-openjpeg](#) – Open source, Python

Image viewers

You can view [image frames](#) after you've decoded them. AWS HealthImaging API actions support a variety of open-source image viewers, including:

- [Open Health Imaging Foundation \(OHIF\)](#)
- [Cornerstone.js](#)

AWS HealthImaging endpoints and quotas

The following topics contain information about AWS HealthImaging service endpoints and quotas.

Topics

- [Service endpoints](#)
- [Service quotas](#)

Service endpoints

A service endpoint is a URL that identifies a host and port as the entry point for a web service. Every web service request contains an endpoint. Most AWS services provide endpoints for specific Regions to enable faster connectivity. The following table lists the service endpoints for AWS HealthImaging.

Region Name	Region	Endpoint	Protocol	
US East (N. Virginia)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol
US West (Oregon)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS

If you are using HTTP requests to call AWS HealthImaging actions, you must use different endpoints depending on the actions being called. The following menu lists the available service endpoints for HTTP requests and the actions they support.

Supported API actions for HTTP requests

data store, import, tagging

The following *data store*, *import*, and *tagging* actions are accessible via endpoint:

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- StartDICOMImportJob
- GetDICOMImportJob

- ListDICOMImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

The following *image set* actions are accessible via endpoint:

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging offers a representations of DICOMweb Retrieve WADO-RS services. For more information, see [Retrieving DICOM data from HealthImaging](#).

The following DICOMweb services are accessible via endpoint:

```
https://dicom-medical-imaging.region.amazonaws.com
```

- GetDICOMInstance
- GetDICOMInstanceMetadata
- GetDICOMInstanceFrames

Service quotas

Service quotas are defined as the maximum value for your resources, actions, and items in your AWS account.

Note

For adjustable quotas, you can request a quota increase using the [Service Quotas console](#). For more information, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

The following table lists the default quotas for AWS HealthImaging.

Name	Default	Adjustable	Description
Maximum concurrent CopyImageSet requests per data store	Each supported Region: 100	Yes	The maximum concurrent CopyImageSet requests per data store in the current AWS Region
Maximum concurrent DeleteImageSet requests per data store	Each supported Region: 100	Yes	The maximum concurrent DeleteImageSet requests per data store in the current AWS Region
Maximum concurrent UpdateImageSetMetadata requests per data store	Each supported Region: 100	Yes	The maximum concurrent UpdateImageSetMetadata requests per data store in the current AWS Region

Name	Default	Adjustable	Description
			store in the current AWS Region
Maximum concurrent import jobs per data store	ap-southeast-2: 20 Each of the other supported Regions: 100	Yes	The maximum number of concurrent import jobs per data store in the current AWS Region
Maximum data stores	Each supported Region: 10	Yes	The maximum number of active data stores in the current AWS Region
Maximum number of ImageFrames allowed to be copied per CopyImageSet request	Each supported Region: 1,000	Yes	The maximum number of ImageFrames allowed to be copied per CopyImageSet request in the current AWS Region
Maximum number of files in a DICOM import job	Each supported Region: 5,000	Yes	The maximum number of files in a DICOM import job in the current AWS Region
Maximum number of nested folders in a DICOM import job	Each supported Region: 10,000	No	The maximum number of nested folders in a DICOM import job in the current AWS Region
Maximum payload size limit (in KB) accepted by UpdateImageSetMetadata	Each supported Region: 10 Kilobytes	Yes	The maximum payload size limit (in KB) accepted by UpdateImageSetMetadata in the current AWS Region

Name	Default	Adjustable	Description
Maximum size (in GB) of all files in a DICOM import job	Each supported Region: 10 Gigabytes	No	The maximum size (in GB) of all files in a DICOM import job in the current AWS Region
Maximum size (in GB) of each DICOM P10 file in a DICOM import job	Each supported Region: 4 Gigabytes	No	The maximum size (in GB) of each DICOM P10 file in the DICOM import job in the current AWS Region
Maximum size limit (in MB) on ImageSetMetadata per Import, Copy, and UpdateImageSet	Each supported Region: 50 Megabytes	Yes	The maximum size limit (in MB) on ImageSetMetadata per Import, Copy, and UpdateImageSet in the current AWS Region

AWS HealthImaging throttling limits

Your AWS account has throttling limits that apply to AWS HealthImaging API actions. For all actions, a `ThrottlingException` error is thrown if throttling limits are exceeded. For more information, see the [AWS HealthImaging API Reference](#).

Note

Throttling limits are adjustable for all HealthImaging API actions. To request a throttling limit adjustment, contact the [AWS Support Center](#). To create a case, log in to your AWS account and choose **Create case**.

The following table lists throttling limits for both [native HealthImaging actions](#) and [representations of DICOMweb services](#).

AWS HealthImaging throttling limits

Action	Throttle rate	Throttle burst
CreateDatastore	0.085 tps	1 tps
GetDatastore	10 tps	20 tps
ListDatastores	5 tps	10 tps
DeleteDatastore	0.085 tps	1 tps
StartDICOMImportJob	0.25 tps	1 tps
GetDICOMImportJob	25 tps	50 tps
ListDICOMImportJobs	10 tps	20 tps
SearchImageSets	25 tps	50 tps
GetImageSet	25 tps	50 tps
GetImageSetMetadata	50 tps	100 tps
GetImageFrame	1000 tps	2000 tps
ListImageSetVersions	25 tps	50 tps
UpdateImageSetMetadata	0.25 tps	1 tps
CopyImageSet	0.25 tps	1 tps
DeleteImageSet	0.25 tps	1 tps
TagResource	10 tps	20 tps
ListTagsForResource	10 tps	20 tps
UntagResource	10 tps	20 tps
GetDICOMInstance*	50 tps	100 tps
GetDICOMInstanceMetadata*	50 tps	100 tps

Action	Throttle rate	Throttle burst
GetDICOMInstanceFrames*	50 tps	100 tps

*Representation of a DICOMweb service

AWS HealthImaging sample projects

AWS HealthImaging provides the following sample projects on GitHub.

[DICOM Ingestion From On-Premises to AWS HealthImaging](#)

An AWS serverless project for deploying an IoT edge solution that receives DICOM files from a DICOM DIMSE source (PACS, VNA, CT scanner) and stores them in a secure Amazon S3 bucket. The solution indexes the DICOM files in a database and queues each DICOM series to be imported in AWS HealthImaging. It is comprised of a component running at the edge that is managed by [AWS IoT Greengrass](#), and a DICOM ingestion pipeline running in AWS Cloud.

[Tile Level Marker \(TLM\) Proxy](#)

An [AWS Cloud Development Kit \(AWS CDK\)](#) project for retrieving image frames from AWS HealthImaging by using tile level markers (TLM), a feature of High-Throughput JPEG 2000 (HTJ2K). This results in faster retrieval times with lower-resolution images. Potential workflows include generating thumbnails and progressive loading of images.

[Amazon CloudFront Delivery](#)

An AWS serverless project for creating an [Amazon CloudFront](#) distribution with an HTTPS endpoint that caches (by using GET) and delivers image frames from the edge. By default, the endpoint authenticates requests with an Amazon Cognito JSON web token (JWT). Both authentication and request signing is done at the edge using [Lambda@Edge](#). This service is a feature of Amazon CloudFront that lets you run code closer to users of your application, which improves performance and reduces latency. There is no infrastructure to manage.

[AWS HealthImaging Viewer UI](#)

An [AWS Amplify](#) project for deploying a frontend UI with backend authentication with which you can view image set metadata attributes and image frames (pixel data) stored in AWS HealthImaging using progressive decoding. You can optionally integrate the Tile Level Marker

(TLM) Proxy and/or Amazon CloudFront Delivery projects above to load image frames using an alternative method.

[AWS HealthImaging DICOMweb Proxy](#)

A Python-based project for enabling DICOMweb WADO-RS and QIDO-RS endpoints on a HealthImaging data store to support web-based medical imaging viewers and other DICOMweb-compatible apps.

Note

This project does not use HealthImaging's representation of DICOMweb APIs described in [Using DICOMweb with AWS HealthImaging](#).

To view additional sample projects, see [AWS HealthImaging Samples](#) on GitHub.

Using HealthImaging with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples

SDK documentation	Code examples
AWS Tools for PowerShell	Tools for PowerShell code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

AWS HealthImaging releases

The following table shows when features and updates were released for the AWS HealthImaging service and documentation. For more information on a release, see the linked topic.

Change	Description	Date
Lossy support for StartDICOMImportJob	HealthImaging supports importing and storing DICOM lossy files (1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50) and binary segmentation files in their original format. For more information, see Supported transfer syntaxes .	November 1, 2024
Lossy support for DICOMweb retrieval services	HealthImaging supports retrieving images and instances stored in 1.2.840.10008.1.2.4.203, 1.2.840.10008.1.2.4.91, 1.2.840.10008.1.2.4.50, and 1.2.840.10008.1.2.1 (binary segmentation only) in their original format or in Explicit VR Little Endian (1.2.840.10008.1.2.1). For more information, see Supported transfer syntaxes and Retrieving DICOM data .	November 1, 2024
Faster imports for digital pathology	HealthImaging supports up to 6x faster import jobs for DICOM digital pathology (WSI).	November 1, 2024

Binary segmentation support	HealthImaging supports both ingestion and retrieval of DICOM binary segmentation files. For more information, see Supported transfer syntaxes .	November 1, 2024
Revert to a previous image set version ID	HealthImaging provides the <code>revertToVersionId</code> parameter to revert to a previous image set version ID. For more information, see revertToVersionId in the <i>AWS HealthImaging API Reference</i> .	July 24, 2024

[Force functionality for image set modification](#)

July 24, 2024

HealthImaging provides the `Overrides` data type with the optional `forced` request parameter. Setting this parameter forces the `UpdateImageSetMeta` data and `CopyImageSet` actions, even if `Patient`, `Study`, or `Series` level metadata are mismatched. For more information, see [Overrides](#) in the *AWS HealthImaging API Reference*.

- `UpdateImageSetMeta` data force functionality—HealthImaging introduces the optional `force` request parameter for updating the following attributes:
 - `Tag.StudyInstanceUID`, `Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID`, and `Tag.StudyID`
 - Adding, removing, or updating instance level private DICOM data elements

For more information, see [UpdateImageSetMetadata](#) in the *AWS HealthImaging API Reference*.

- `CopyImageSet` force functionality—Heal

HealthImaging introduces the optional `forceRequest` parameter for copying image sets. Setting this parameter forces the `CopyImageSet` action, even if Patient, Study, or Series level metadata are mismatched across the `sourceImageSet` and `destinationImageSet`. In these cases, the inconsistent metadata remains unchanged in the `destinationImageSet`. For more information, see [CopyImageSet](#) in the *AWS HealthImaging API Reference*.

[Copy subsets of SOP Instances](#)

HealthImaging enhances the `CopyImageSet` action so you can pick one or more SOP Instances from a `sourceImageSet` to copy to a `destinationImageSet`. For more information, see [Copying an image set](#).

July 24, 2024

[GetDICOMInstanceMetadata for returning DICOM instance metadata](#)

HealthImaging provides the `GetDICOMInstanceMetadata` service to return DICOM Part 10 metadata (.json file). For more information, see [Getting instance metadata](#).

July 11, 2024

[GetDICOMInstanceFrames for returning DICOM instance frames \(pixel data\)](#)

HealthImaging provides the GetDICOMInstanceFrames service to return DICOM Part 10 frames (multipart request). For more information, see [Getting instance frames](#).

July 11, 2024

[Enhanced support for non-standard DICOM data imports](#)

June 28, 2024

HealthImaging provides support for data imports that include deviations from the DICOM standard. For more information, see [DICOM element constraints](#).

- The following DICOM data elements can be up to 256 characters in max length:
 - Patient's Name
(0010,0010)
 - Patient ID
(0010,0020)
 - Accession Number
(0008,0050)
- The following syntax variations are permitted for Study Instance UID, Series Instance UID, Treatment Session UID, Manufacturer's Device Class UID, Device UID, and Acquisition UID :
 - The first element of any UID can be zero
 - UIDs can start with one or more leading zeros
 - UIDs can be up to 256 characters in length

[Event notifications](#)

HealthImaging integrates with Amazon EventBridge to support event-driven applications. For more information, see [Using EventBridge](#).

June 5, 2024

[GetDICOMInstance for returning DICOM instance data](#)

HealthImaging provides the GetDICOMInstance service to return DICOM Part 10 instance data (.dcm file). For more information, see [Getting an instance](#).

May 15, 2024

[Cross-account import](#)

HealthImaging provides support for data imports from Amazon S3 buckets located in other supported Regions. For more information, see [Cross-account import](#).

May 15, 2024

[Search enhancements for image sets](#)

HealthImaging SearchImageSets action supports the following search enhancements. For more information, see [Searching image sets](#).

April 3, 2024

- Additional support for searching on UpdatedAt and SeriesInstanceUID
- Search between start time and end time
- Sort search results by Ascending or Descending
- DICOM Series parameters are returned in responses

[Maximum file size for imports increased](#)

HealthImaging supports a 4 GB maximum file size for each DICOM P10 file in an import job. For more information, see [Service quotas](#).

March 6, 2024

[Transfer syntaxes for JPEG Lossless and HTJ2K](#)

HealthImaging provides support for the following transfer syntaxes for job imports. For more information, see [Supported transfer syntaxes](#).

February 16, 2024

- 1.2.840.10008.1.2.4.57 — JPEG Lossless Non-Hierarchical (Process 14)
- 1.2.840.10008.1.2.4.201 — High-Throughput JPEG 2000 Image Compression (Lossless Only)
- 1.2.840.10008.1.2.4.202 — High-Throughput JPEG 2000 with RPCL Options Image Compression (Lossless Only)
- 1.2.840.10008.1.2.4.203 — High-Throughput JPEG 2000 Image Compression

[Tested code examples](#)

HealthImaging documentation provides tested code examples for AWS CLI and AWS SDKs for Python, JavaScript, Java, and C++. For more information, see [Code examples](#).

December 19, 2023

[Maximum file number for imports increased](#)

HealthImaging supports up to 5,000 files for a single import job. For more information, see [Service quotas](#).

December 19, 2023

Nested folders for imports	HealthImaging supports up to 10,000 nested folders for a single import job. For more information see Service quotas .	December 1, 2023
Faster imports	HealthImaging provides 20X faster imports in all supported Regions. For more information, see Service endpoints .	December 1, 2023
AWS CloudFormation support	HealthImaging supports infrastructure as code (IaC) for provisioning data stores. For more information, see Creating HealthImaging resources with AWS CloudFormation .	September 21, 2023
General availability	AWS HealthImaging is available to all customers in the US East (N. Virginia), US West (Oregon), Europe (Ireland), and Asia Pacific (Sydney) Regions. For more information, see Service endpoints .	July 26, 2023