aws

开发人员指南

AWS HealthImaging

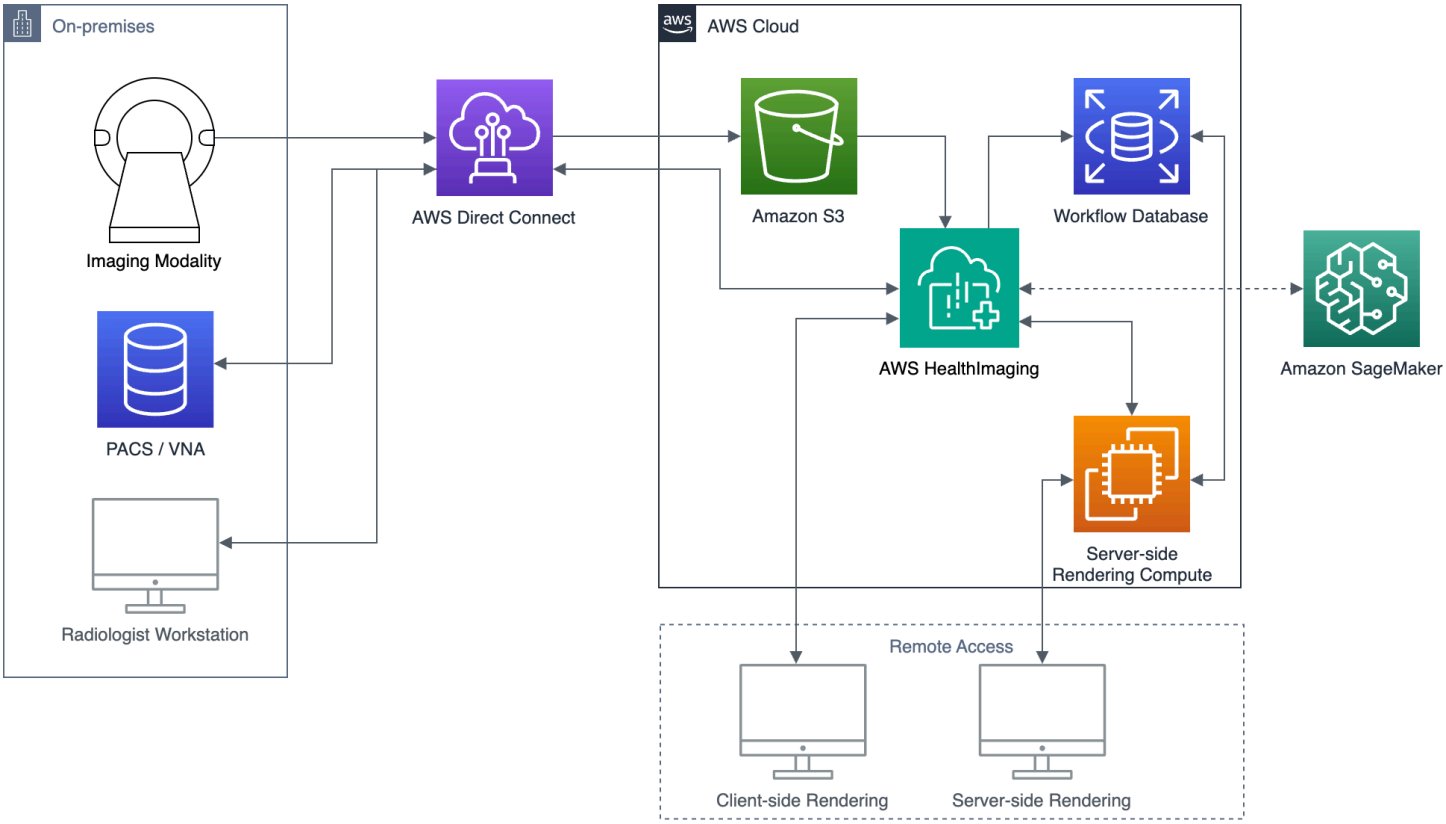
AWS HealthImaging: 开发人员指南

Table of Contents

什么是 AWS HealthImaging？

AWS HealthImaging 是一项符合 HIPAA 资格的服务，它使医疗保健提供商、生命科学组织及其软件合作伙伴能够在 PB 级的云中存储、分析和共享医疗图像。 HealthImaging用例包括：

- 企业成像-直接从 AWS 云端存储和流式传输您的医疗成像数据，同时保持低延迟性能和高可用性。
- 长期图像存档-节省长期图像存档的成本，同时保持亚秒级的图像检索访问权限。
- AI/ML 开发 — 在其他工具和服务的支持下，对图像存档运行人工智能和机器学习 (AI/ML) 推理。
- 多模态分析 — 将您的临床成像数据与 AWS HealthLake （健康数据）和 AWS HealthOmics （组学数据）相结合，为精准医疗提供见解。



AWS HealthImaging 提供对图像数据（例如 X-Ray、CT、MRI、超声）的访问权限，因此云中内置的医学成像应用程序可以实现以前只能在本地才能实现的性能。借 HealthImaging助，您可以从中每张医学图像的单一权威副本大规模运行医学成像应用程序，从而降低基础设施成本 AWS Cloud。

主题

重要提示

HealthImaging AWS 不能替代专业医疗建议、诊断或治疗，也不能用于治愈、治疗、缓解、预防或诊断任何疾病或健康状况。您有责任在 AWS 的任何使用中进行人工审查 HealthImaging，包括与任何旨在为临床决策提供依据的第三方产品相关的审查。只有经过培训的医疗专业人员根据合理的医学判断进行审核后，才 HealthImaging 应将 AWS 用于患者护理或临床场景。

AWS 的特点 HealthImaging

AWS HealthImaging 提供以下功能。

开发人员友好的 DICOM 元数据

AWS 以开发人员友好的格式返回 DICOM 元数据， HealthImaging 从而简化了应用程序开发。导入影像数据后，可以使用人性化的关键词来访问各个元数据属性，而不是使用不熟悉的组/元素十六进制数字。患者、研究和系列层面的 DICOM 元素已标准化，无需应用程序开发人员处理 SOP 实例之间的不一致之处。此外，可以在本机运行时系统类型中直接访问元数据属性值。

SIMD 加速影像解码

AWS HealthImaging 返回使用高吞吐量 JPEG 2000 (HTJ2K)（一种高级图像压缩编解码器）编码的图像帧（像素数据）。HTJ2K 利用现代处理器上的单指令多数据 (SIMD，Single Instruction Multiple Data) 来提供更高的性能。HTJ2K 比 JPEG2000 快一个数量级，比所有其他 DICOM 传输语法至少快一倍。可以利用 WASM-SIMD 为零足迹的网络浏览者带来这种极快的速度。

像素数据验证

AWS 通过在导入过程中检查每张图像的无损编码和解码状态来 HealthImaging 提供内置的像素数据验证。有关更多信息，请参阅 像素数据验证。

业界领先的性能

AWS 凭借其高效的元数据编码、无损压缩和渐进式分辨率数据访问，为图像加载性能 HealthImaging 树立了新的标准。高效的元数据编码使能影像查看器和 AI 算法无需加载影像数据即

可理解 DICOM 研究的内容。得益于先进的图像压缩技术，图像加载速度更快，图像质量却丝毫不受影响。递增分辨率可以更快地加载缩略图、感兴趣区域和低分辨率移动设备的影像。

可扩展的 DICOM 导入

AWS HealthImaging 导入利用现代云原生技术并行导入多个 DICOM 研究。可以快速导入历史档案，而不会影响新数据的临床工作负载。有关支持的 SOP 实例和传输语法的信息，请参阅 DICOM 支持。

相关 AWS 服务

AWS HealthImaging 与其他 AWS 服务紧密集成。了解以下服务对于充分利用很有用 HealthImaging。

- AWS Identity and Access Management— 您可以使用 IAM 来安全地管理身份和对 HealthImaging 资源的访问权限。
- 亚马逊简单存储服务 — 使用 Amazon S3 作为暂存区，将 DICOM 数据导入其中。 HealthImaging
- Amazon CloudWatch — 您 CloudWatch 用来观察和监控 HealthImaging 资源。
- AWS CloudTrail— 您 CloudTrail 用来跟踪 HealthImaging 用户活动和 API 使用情况。
- AWS CloudFormation— 您可以使用 AWS CloudFormation 实现基础设施即代码 (IaC) 模板来创建资源。 HealthImaging
- AWS PrivateLink— 您可以使用 Amazon VPC 在 HealthImaging 和亚马逊 Virtual Private Cloud 之间建立连接，而无需将数据暴露给互联网。

访问 AWS HealthImaging

您可以 HealthImaging 使用 AWS Management Console、 AWS Command Line Interface 和 AWS 软件开发工具包访问 AWS。本指南提供了程序说明， AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。

AWS Management Console

AWS Management Console 提供了一个基于 Web 的用户界面，用于管理 HealthImaging 及其相关资源。如果您已注册 AWS 帐户，则可以登录HealthImaging 控制台。

AWS Command Line Interface (AWS CLI)

AWS CLI 提供了适用于各种 AWS 产品的命令，并在 Windows、Mac 和 Linux 上受支持。有关更多信息，请参阅《AWS Command Line Interface 用户指南》。

AWS 软件开发工具包

AWS SDK 为软件开发人员提供库、代码示例和其他资源。这些库文件提供可自动执行任务的基本功能，例如以加密方式对请求签名、重试请求和处理错误响应。有关更多信息，请参阅用于在AWS上进行构建的工具。

HTTP 请求

您可以使用 HTTP 请求调用 HealthImaging 操作，但必须根据所使用的操作类型指定不同的终端节点。有关更多信息，请参阅 HTTP 请求支持的 API 操作。

HIPAA 资格和数据安全

这是一项符合 HIPAA 要求的服务。有关 AWS《1996 年美国健康保险流通与责任法案》(HIPAA) 以及使用 AWS 服务处理、存储和传输受保护的健康信息 (PHI) 的更多信息，请参阅 HIPAA 概述。

必须对与 HealthImaging 包含 PHI 和个人身份信息 (PII) 的连接进行加密。默认情况下，所有连接都 HealthImaging 使用通过 TLS 的 HTTPS。 HealthImaging 存储加密的客户内容，并按照责任AWS 共担模式运营。

有关合规性的信息，请参阅AWS HealthImaging 的合规性验证。

定价

HealthImaging 通过智能分层帮助您实现临床数据的生命周期管理自动化。有关更多信息，请参阅 了解存储层。

有关一般定价信息，请参阅 AWS HealthImaging 定价。要估算成本，请使用 AWS HealthImaging 定价计算器。

开始使用 AWS HealthImaging

要开始使用 AWS HealthImaging，请设置 AWS 账户并创建 AWS Identity and Access Management 用户。要使用 AWS CLI 或 AWS SDK，必须安装并对其进行配置。

在学习了 HealthImaging 概念和设置之后，我们提供了一个包含代码示例的简短教程来帮助您入门。

主题

AWS HealthImaging 概念

以下术语和概念对您了解和使用 AWS HealthImaging 非常有用。

概念

数据存储

数据存储是驻留在单个 AWS 区域 中的医学影像数据存储库。一个 AWS 账户可以有零个或多个数据存储。数据存储有自己的 AWS KMS 加密密钥，因此一个数据存储中的数据可以在物理和逻辑上与其他数据存储中的数据隔离。数据存储支持使用 IAM 角色、权限和基于属性的访问控制进行访问控制。

有关更多信息，请参阅 管理数据存储 和 了解存储层：

影像集

影像集是一个 AWS 概念，它定义了一种用于优化相关医学影像数据的抽象分组机制。当您将 DICOM P10 影像数据导入 AWS HealthImaging 数据存储时，它会转换为由元数据和图像帧（像素数据）组成的影像集。导入 DICOM P10 数据会生成包含同一 DICOM 系列中一个或多个服务对象对 (SOP, Service-Object Pair) 实例的 DICOM 元数据和影像帧的影像集。

有关更多信息，请参阅 导入影像数据 和 了解图像集：

元数据

元数据是影像集中存在的非像素属性。对于 DICOM 来说，这包括患者人口统计、手术细节和其他特定采集参数。AWS HealthImaging 将影像集分成元数据和图像帧（像素数据），因此应用程序可以快速访问它。这对于不需要像素数据的影像查看器、分析和 AI/ML 用例非常有用。DICOM 数据在患者、研究和系列级别进行标准化，从而消除了不一致之处。这简化了数据的使用，提高了安全性，并提高了访问性能。

有关更多信息，请参阅 获取影像集元数据 和 元数据标准化：

影像帧

影像帧是影像集中存在的用于构成 2D 医学影像的像素数据。在导入过程中，AWS HealthImaging 以高吞吐量 JPEG 2000 (HTJ2K) 对所有影像帧进行编码。因此，必须先对影像帧进行解码才能观看。

有关更多信息，请参阅 获取影像集像素数据 和 HTJ2K 解码库：

设置 AWS HealthImaging

在使用 AWS 之前，您必须设置您的 AWS 环境 HealthImaging。以下主题是下一节中教程的先决条件。

主题

- 注册获取 AWS 账户
- 创建具有管理访问权限的用户
- 创建 S3 存储桶
- 创建数据存储
- 创建具有 HealthImaging 完全访问权限的 IAM 用户
- 为导入创建 IAM 角色
- 安装 AWS CLI（可选）

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。
2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

创建 S3 存储桶

要将 DICOM P10 数据导入 AWS HealthImaging，建议使用两个 Amazon S3 存储桶。Amazon S3 输入存储桶存储要导入并从该存储桶 HealthImaging 读取的 DICOM P10 数据。Amazon S3 输出存储桶存储导入任务的处理结果并 HealthImaging写入该存储桶。有关此内容的直观展示，请参阅位于 了解导入任务 的示意图。

> **Note**
>
> 根据 AWS Identity and Access Management (IAM) 政策，您的 Amazon S3 存储桶名称必须是唯一的。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的存储桶命名规则。

出于本指南的目的，我们在 IAM 角色中指定以下 Amazon S3 输入和输出存储桶进行导入。

- 输入存储桶: `arn:aws:s3:::medical-imaging-dicom-input`
- 输出桶：`arn:aws:s3:::medical-imaging-output`

有关更多信息，请参阅《Amazon S3 用户指南》中的创建存储桶。

创建数据存储

当您导入医学影像数据时，AWS HealthImaging 数据存储会保存转换后的 DICOM P10 文件（称为图像集）的结果。有关此内容的直观展示，请参阅位于 了解导入任务 的示意图。

> **Tip**
>
> 创建数据存储时会生成 `datastoreID`。在本节后面完成导入 trust relationship 时，必须使用 `datastoreID`。

要创建数据存储，请参阅 创建数据存储。

创建具有 HealthImaging 完全访问权限的 IAM 用户

> **最佳实践**
>
> 我们建议您创建单独的 IAM 用户，以满足不同的需求，例如导入、数据访问和数据管理。这与 AWS 架构完善的框架中的授予最低访问权限一致。
> 就下一节的教程而言，您将使用单个 IAM 用户。

若要创建 IAM 用户

1. 按照 IAM 用户指南中有关在您的 AWS 账户中创建 IAM 用户的说明进行操作。为澄清起见，请考虑命名用户为 `ahiadmin`（或类似名称）。
2. 将托管的 IAM 策略 `AWSHealthImagingFullAccess` 分配给 IAM 用户。有关更多信息，请参阅 AWS 托管策略：AWSHealthImagingFullAccess。

   > **Note**
   >
   > 可以缩小 IAM 权限的范围。有关更多信息，请参阅 适用于 AWS HealthImaging 的 AWS 托管策略。

为导入创建 IAM 角色

> **Note**
>
> 以下说明涉及一个 AWS Identity and Access Management (IAM) 角色，该角色授予对 Amazon S3 存储桶的读取和写入权限，以导入您的 DICOM 数据。尽管下一节的教程需要该角色，但我们建议您使用 适用于 AWS HealthImaging 的 AWS 托管策略 为用户、群组和角色添加 IAM 权限，因为使用它们比自己编写策略更容易。

IAM 角色是可在账户中创建的一种具有特定权限的 IAM 身份。要启动导入任务，必须将调用 `StartDICOMImportJob` 操作的 IAM 角色附加到用户策略，该策略允许访问用于读取 DICOM P10 数据和存储导入任务处理结果的 Amazon S3 存储桶。还必须为其分配信任关系（策略），使 AWS HealthImaging 能够担任该角色。

为导入创建 IAM 角色

1. 使用 IAM 控制台，创建名为 `ImportJobDataAccessRole` 的角色。您将在下一节的教程中使用此角色。有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 角色。

   > **Tip**
   >
   > 就本指南而言，启动导入任务 中的代码示例引用了 `ImportJobDataAccessRole` IAM 角色。

2. 对于 IAM 角色附加 IAM 权限策略。此权限策略授予对 Amazon S3 输入和输出存储桶的访问权限。将以下权限策略附加到此 IAM 角色 `ImportJobDataAccessRole`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::medical-imaging-dicom-input",
                "arn:aws:s3:::medical-imaging-output"
            ],
            "Effect": "Allow"
```

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

3. 将以下信任关系（策略）附加到 ImportJobDataAccessRole IAM 角色。信托策略需要您在完成第 创建数据存储 部分时生成的 datastoreId。本主题之后的教程假设您使用的是一个 AWS HealthImaging 数据存储，但使用的是特定于数据存储的 Amazon S3 存储桶、IAM 角色和信任策略。

```
{
    "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"
```

```
                }
            }
        }
    ]
}
```

要了解有关在 AWS 中创建和使用 IAM 策略的更多信息 HealthImaging，请参阅适用于 AWS 的 Identity and Access 管理 HealthImaging。

有关 IAM 角色的更多一般信息，请参阅《IAM 用户指南》中的 IAM 角色。有关 IAM Policy 的更多一般信息，请参阅《IAM 用户指南》中的 IAM 策略与权限。

安装 AWS CLI（可选）

如果您使用的是 AWS Command Line Interface，则需要执行以下步骤。如果您使用的是 AWS Management Console 或 S AWS DK，则可以跳过以下步骤。

要设置 AWS CLI

1. 下载并配置 AWS CLI。有关说明，请参阅AWS Command Line Interface 用户指南中的以下主题。

   - 安装或更新最新版本的 AWS CLI
   - 开始使用 AWS CLI

2. 在 AWS CLI `config`文件中，为管理员添加已命名的配置文件。运行 AWS CLI 命令时使用此配置文件。根据最低权限的安全原则，我们建议您创建一个单独的 IAM 角色，该角色具有特定于正在执行的任务的权限。有关已命名配置文件的更多信息，请参阅《AWS Command Line Interface 用户指南》中的配置和凭证文件设置。

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

3. 请使用以下 `help` 命令验证设置：

   ```
   aws medical-imaging help
   ```

   如果配置 AWS CLI 正确，您将看到 AWS 的简要描述 HealthImaging和可用命令列表。

AWS HealthImaging 教程

目标

本教程的目标是将 DICOM P10 文件导入 AWS HealthImaging 数据存储，并将其转换为由元数据和图像框（像素数据）组成的图像集。导入 DICOM 数据后，您可以根据对的访问偏好访问影像集、元数据和图像框。 HealthImaging

先决条件

设置 中列出的所有步骤都是完成本教程所必需的。

教程步骤

1. 开始导入任务
2. 获取导入任务属性
3. 搜索影像集
4. 获取影像集属性
5. 获取影像集元数据
6. 获取影像集像素数据
7. 删除数据存储

使用 AWS 管理数据存储 HealthImaging

使用 AWS HealthImaging，您可以创建和管理医疗图像资源的数据存储。以下主题介绍如何使用、和 AWS SDK 使用 HealthImaging 操作创建、描述、列出和删除数据存储。 AWS Management Console AWS CLI

> **Note**
>
> 本章最后一个主题是了解存储层。将医学影像数据导入数据存储后，数据会根据时间和使用情况自动在两个存储层之间移动。存储层具有不同的定价级别，因此了解存储层移动过程以及用于计费目的的 HealthImaging 资源非常重要。

主题

创建数据存储

您可以使用`CreateDatastore`操作创建用于导入 DICOM P10 文件的 AWS HealthImaging 数据存储。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`CreateDatastore`*中的。

> **重要提示**
>
> 请勿使用受保护的健康信息 (PHI)、个人身份信息 (PII) 或其他机密或敏感信息为数据存储命名。

创建数据存储

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台的 “创建数据存储” 页面。
2. 在详细信息下的数据存储名称中，输入数据存储的名称。
3. 在 “数据加密” 下，选择用于加密资源的 AWS KMS 密钥。有关更多信息，请参阅 AWS 中的数据保护 HealthImaging。
4. 在标签 - 可选下，您可以在创建数据存储时向其添加标签。有关更多信息，请参阅 标记资源。
5. 选择创建数据存储。

AWS CLI 和 SDK

Bash

AWS CLI 使用 Bash 脚本

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

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

```
  local datastore_name response
  local option OPTARG # Required to use getopts command in a function.

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

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

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

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

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports medical-imaging create-datastore operation
 failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

创建数据存储

以下 `create-datastore` 代码示例创建名称为 `my-datastore` 的数据存储。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的创建数据存储。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateDatastore中的。

Java

适用于 Java 2.x 的 SDK

```
   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 "";
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

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

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考CreateDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用CreateDatastore于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取数据存储属性

您可以使用`GetDatastore`操作来检索 AWS HealthImaging 数据存储的属性。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`GetDatastore`*中的。

获取数据存储属性

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开。在详细信息部分下，列出了所有可用的数据存储属性。要查看关联的图像集、导入图像和标签，请选择相应的选项卡。

AWS CLI 和 SDK

Bash

AWS CLI 使用 Bash 脚本

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

##############################################################################
# function imaging_get_datastore
#
# Get a data store's properties.
#
# Parameters:
#       -i data_store_id - The ID of the data store.
#
# Returns:
#       [datastore_name, datastore_id, datastore_status, datastore_arn,
 created_at, updated_at]
#    And:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function imaging_get_datastore() {
```

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

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

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

 local response

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

```
  )
  error_code=${?}

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

  echo "$response"

  return 0
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

获取数据存储的属性

以下 `get-datastore` 代码示例可获取数据存储的属性。

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

输出：

```
{
    "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"
    }
}
```

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的获取数据存储属性。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetDatastore中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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"];
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

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

- 有关 API 的详细信息，请参阅适用GetDatastore于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出数据存储

您可以使用`ListDatastores`操作列出 AWS 中的可用数据存储 HealthImaging。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*ListDatastores*中的。

列出数据存储

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

- 打开 HealthImaging 控制台数据存储页面。

  所有数据存储都列在数据存储部分下。

AWS CLI 和 SDK

Bash

AWS CLI 使用 Bash 脚本

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

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

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

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

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

  echo "$response"

  return 0
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

列出数据存储

以下 `list-datastores` 代码示例列出可用的数据存储。

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

输出：

```
{
    "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"
```

```
        }
    ]
}
```

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的列出数据存储。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListDatastores中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

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

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
```

```
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用ListDatastores于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除数据存储

您可以使用`DeleteDatastore`操作删除 AWS HealthImaging 数据存储。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`DeleteDatastore`*中的。

> **Note**
>
> 在删除数据存储之前，必须先删除其中的所有图像集。有关更多信息，请参阅 删除一个影像集。

删除数据存储

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。
3. 选择 删除。

   删除数据存储页面将会打开。
4. 要确认删除数据存储，请在文本输入字段中输入数据存储名称。
5. 选择删除数据存储。

AWS CLI 和 SDK

Bash

AWS CLI 使用 Bash 脚本

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

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

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

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

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

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

  local error_code=${?}

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

  return 0
```

```
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

删除数据存储

以下 `delete-datastore` 代码示例可删除数据存储。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的删除数据存储。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteDatastore中的。

Java

适用于 Java 2.x 的 SDK

```
   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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

```
  //           totalRetryDelay: 0
  //         },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用DeleteDatastore于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

了解存储层

AWS HealthImaging 使用智能分层进行自动临床生命周期管理。这为新数据或活动数据以及长期存档数据带来了颇具吸引力的性能和价格，而且不会产生任何摩擦。HealthImaging 使用以下层按 GB/月计费存储费用。

- 频繁访问层 – 频繁访问的数据的层。
- 存档即时访问层 – 存档数据的层。

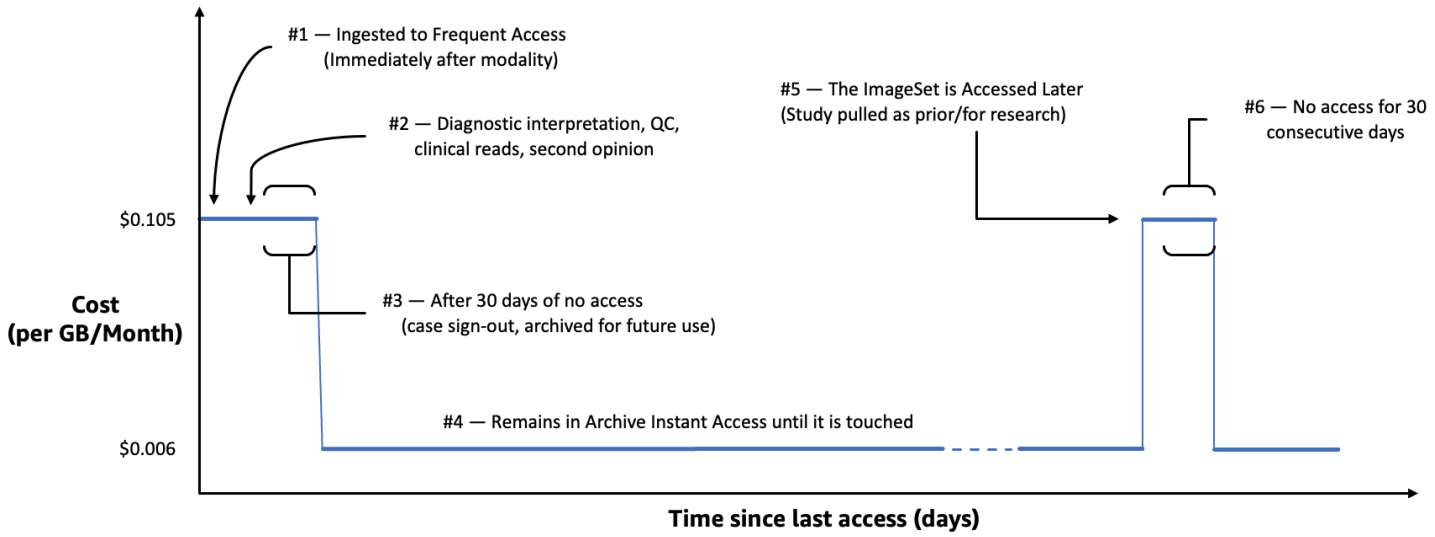
> **Note**
>
> 频繁访问层 和存档即时访问层之间没有性能差异。智能分层应用于特定的图像集 API 操作。智能分层无法识别数据存储、导入和标记 API 操作。各层之间的移动是根据 API 使用情况自动进行的，这将在下一节中进行说明。

层移动是如何运作的？

- 导入后，图像集从频繁访问层开始。

- 在连续 30 天无接触后，图像集会自动移至存档即时访问层。
- 存档即时访问层中的图像集只有在受到接触后才会返回到频繁访问层。

下图概述了 HealthImaging 智能分层流程。



什么才算是接触？

接触是通过 AWS Management Console、AWS CLI 或 AWS 软件开发工具包访问特定的 API ，发生在以下情况下：

1. 已创建新的图像集（`StartDICOMImportJob` 或 `CopyImageSet`）
2. 图像集已更新（`UpdateImageSetMetadata` 或 `CopyImageSet`）
3. 读取图像集的关联元数据或图像帧（像素数据）（`GetImageSetMetaData` 或 `GetImageFrame`）

以下 HealthImaging API 操作会接触图像集并将其从存档即时访问层移动到频繁访问层。

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

> **Note**
>
> 尽管无法使用 UpdateImageSetMetadata 操作删除图像帧（像素数据），但出于计费目的，它们仍会被计算在内。

以下 HealthImaging API 操作不会导致接触。因此，此类操作不会将图像集从存档即时访问层移动到频繁访问层。

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

使用 AWS HealthImaging 导入影像数据

导入是将您的医学影像数据从 Amazon S3 输入桶移动到 AWS HealthImaging 数据存储的过程。在导入过程中，AWS HealthImaging 会先执行像素数据验证检查，然后再将您的 DICOM P10 文件转换为由元数据和图像框（像素数据）组成的图像集。

> **Tip**
>
> 在您熟悉 HealthImaging 之后，我们鼓励您访问 AWS HealthImaging 示例项目，以使用我们的导入和查看项目快速开始实施。

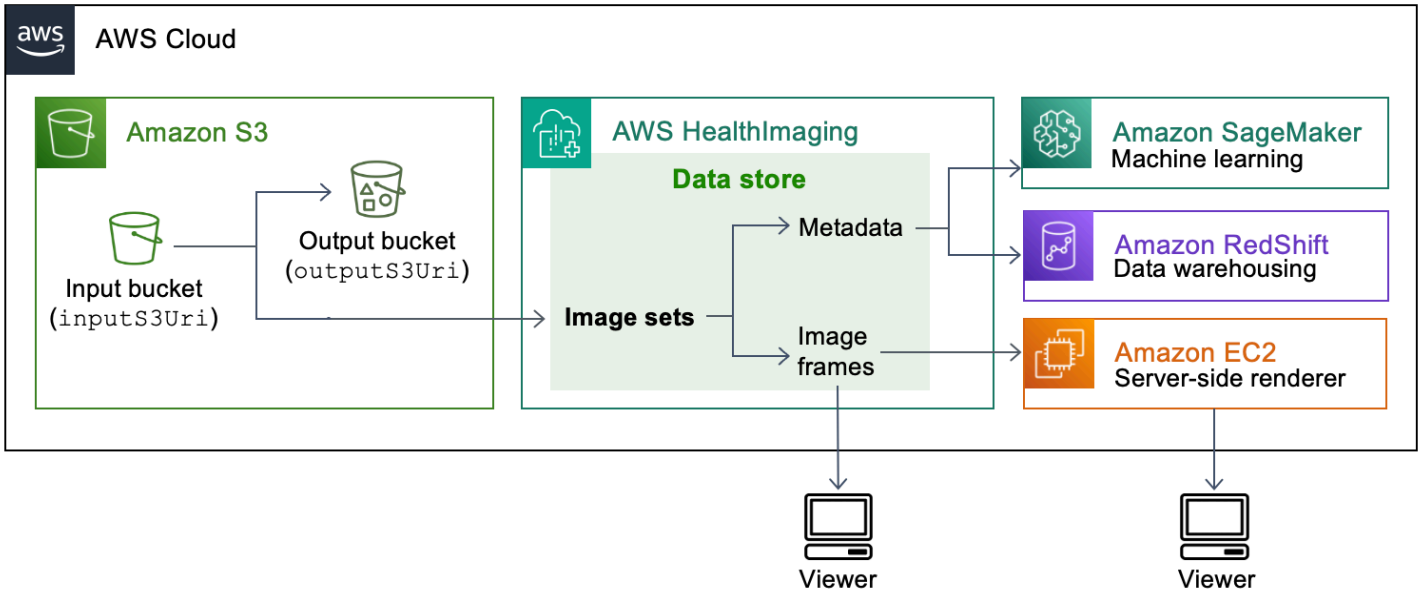
以下主题介绍如何使用 AWS Management Console、AWS CLI 和 AWSSDK 将您的医学影像数据导入到 HealthImaging 数据存储中。

主题

了解导入任务

在 AWS 中创建数据存储后 HealthImaging，您必须将医学影像数据从 Amazon S3 输入存储桶导入数据存储以创建图像集。您可以使用 AWS Management Console AWS CLI、和 AWS SDK 来启动、描述和列出导入任务。

下图概述了如何将 DICOM 数据 HealthImaging 导入数据存储并将其转换为影像集。导入任务处理结果存储在 Amazon S3 输出存储桶 (`outputS3Uri`) 中，图像集存储在 AWS HealthImaging 数据存储中。



将您的医学影像文件从 Amazon S3 导入到 AWS HealthImaging 数据存储时，请记住以下几点：

- 导入任务支持特定的 SOP 类别和传输语法。有关更多信息，请参阅 DICOM 支持。
- 在导入过程中，长度限制适用于特定的 DICOM 元素。为确保成功完成导入任务，请确认您的医学影像数据未超过长度限制。有关更多信息，请参阅 DICOM 元素限制。
- 在导入任务开始时执行像素数据验证检查。有关更多信息，请参阅 像素数据验证。
- 有与 HealthImaging 导入操作相关的终端节点、配额和限制限制。有关更多信息，请参阅 端点和限额 和 节流限制。
- 对于每个导入任务，处理结果都存储在 `outputS3Uri` 位置。处理结果按 `job-output-manifest.json` 文件以及 `SUCCESS` 和 `FAILURE` 文件夹进行组织。

> **Note**
>
> 单个导入任务最多可以包含 10,000 个嵌套文件夹。

- 该 `job-output-manifest.json` 文件包含有关已处理数据的 `jobSummary` 输出和其他详细信息。以下示例显示从 `job-output-manifest.json` 文件的输出。

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

- 该 SUCCESS 文件夹包含所有成功导入的影像文件结果的 success.ndjson 文件。以下示例显示从 success.ndjson 文件的输出。

```
{"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"}}
```

- 该 FAILURE 文件夹包含所有未成功导入的影像文件结果的 failure.ndjson 文件。以下示例显示从 failure.ndjson 文件的输出。

```
{"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"}}
```

- 导入任务将在任务列表中保留 90 天，然后存档。

启动导入任务

您可以使用该`StartDICOMImportJob`操作开始像素数据验证检查，并将数据批量导入到 AWS HealthImaging 数据存储中。导入任务导入位于 `inputS3Uri` 参数指定的 Amazon S3 输入存储桶中的 DICOM P10 文件。导入任务处理结果存储在 `outputS3Uri` 参数指定的 Amazon S3 输出存储桶中。

> **Note**
>
> HealthImaging 支持从位于其他支持区域的 Amazon S3 存储桶导入数据。要实现此功能，请在启动导入任务时提供`inputOwnerAccountId`参数。有关更多信息，请参阅 的跨账户导入 AWS HealthImaging。
> 在导入过程中，长度限制适用于特定的 DICOM 元素。有关更多信息，请参阅 DICOM 元素限制。

以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`StartDICOMImportJob`*中的。

启动导入任务

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。
3. 选择导入 DICOM 数据。

   将打开导入 DICOM 数据页面。
4. 在 “详细信息” 部分下，输入以下信息：

   - 姓名（可选）

- 在 S3 中导入源位置
- 源存储桶所有者的账户 ID（可选）
- 加密密钥（可选）
- S3 中的输出目的地

5. 在服务访问权限一节下，选择使用现有服务角色，然后从服务角色名称菜单中选择该角色或者选择创建并使用新的服务角色。
6. 选择 Import（导入）。

AWS CLI 和 SDK

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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ AP I 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

启动 DICOM 导入任务

以下 start-dicom-import-job 代码示例启动 DICOM 导入任务。

```
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"
```

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的启动导入任务。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》ImportJob中的 StartDicom。

Java

适用于 Java 2.x 的 SDK

```
   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 "";
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x AP I 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input
 files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files
 are stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
```

```
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅 Python AWS SDK (Boto3) API 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取导入任务属性

您可以使用该`GetDICOMImportJob`操作来了解有关 AWS HealthImaging 导入任务属性的更多信息。例如，启动导入任务后，您可以运行 `GetDICOMImportJob` 以查找该作业的状态。一旦 `jobStatus` 返回为 `COMPLETED`，您就可以访问您的影像集了。

> **Note**
>
> `jobStatus` 是指导入作业的执行。因此，即使在导入过程中发现了验证问题，导入任务也可能返回 `jobStatus` 为 `COMPLETED`。如果 `jobStatus` 返回为 `COMPLETED`，我们仍然建议您查看写入 Amazon S3 的输出清单，因为它们提供了有关单个 P10 对象导入成功或失败的详细信息。

以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*GetDICOMImportJob*中的。

如要获取导入任务属性

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

数据存储详细信息页面将会打开。默认情况下，影像集选项卡处于选中状态。

3. 选择导入选项卡。
4. 选择一个导入任务。

将打开导入任务详细信息页面，并显示有关导入任务的属性。

AWS CLI 和 SDK

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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考ImportJob中的 getDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

获取 DICOM 导入任务的属性

以下 `get-dicom-import-job` 代码示例可获取导入任务的属性。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的获取导入任务属性。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》ImportJob中的 getDiCom。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x AP I 参考ImportJob中的 getDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ImportJob中的 getDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

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

- 有关 API 的详细信息，请参阅 Pyt hon AWS SDK (Boto3) API 参考ImportJob中的 getDiC om

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出导入作业

您可以使用`ListDICOMImportJobs`操作列出为特定 HealthImaging 数据存储创建的导入任务。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`ListDICOMImportJobs`*中的。

> **Note**
>
> 导入任务将在任务列表中保留 90 天，然后存档。

列出导入作业

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开。默认情况下，影像集选项卡处于选中状态。
3. 选择导入选项卡以列出所有关联的导入任务。

AWS CLI 和 SDK

CLI

AWS CLI

列出 DICOM 导入任务

以下 `list-dicom-import-jobs` 代码示例列出 DICOM 导入任务。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的列出导入任务。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》ImportJobs中的 ListDicom。

Java

适用于 Java 2.x 的 SDK

```
    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<>();
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x AP I 参考ImportJobs中的 ListDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

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

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

  return jobSummaries;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ImportJobs中的 ListDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用于 Python 的AWS SDK (Boto3) API 参考ImportJobs中的 ListDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 AWS 访问图像集 HealthImaging

在 AWS 中访问医学影像数据 HealthImaging 通常包括使用唯一密钥搜索图像集并获取相关的元数据和图像框架（像素数据）。

> **Tip**
>
> 在您熟悉 AWS 之后 HealthImaging，我们鼓励您AWS HealthImaging 示例项目访问我们的查看项目，快速开始实施。

以下主题说明了什么是影像集，以及如何使用 AWS Management Console AWS CLI、和 AWS SDK 搜索它们并获取其关联的属性、元数据和图像框。

主题

了解图像集

图像集是一个 AWS 构成 AWS 基础的概念 HealthImaging。图像集是在您将 DICOM 数据导入时创建的 HealthImaging，因此在使用该服务时需要对它们有很好的了解。

引入图像集的原因如下：

- 通过灵活的 API 支持各种医学影像工作流程（临床和非临床）。
- 通过仅对相关数据进行分组，最大限度地提高患者安全。
- 鼓励清理数据，以帮助提高不一致性的可见性。有关更多信息，请参阅 修改图像集。

> **重要提示**
>
> 在清理 DICOM 数据之前对其进行临床使用，可能会对患者造成伤害。

以下菜单进一步详细描述了影像集，并提供了示例和图表，以帮助您理解其功能和用途。HealthImaging

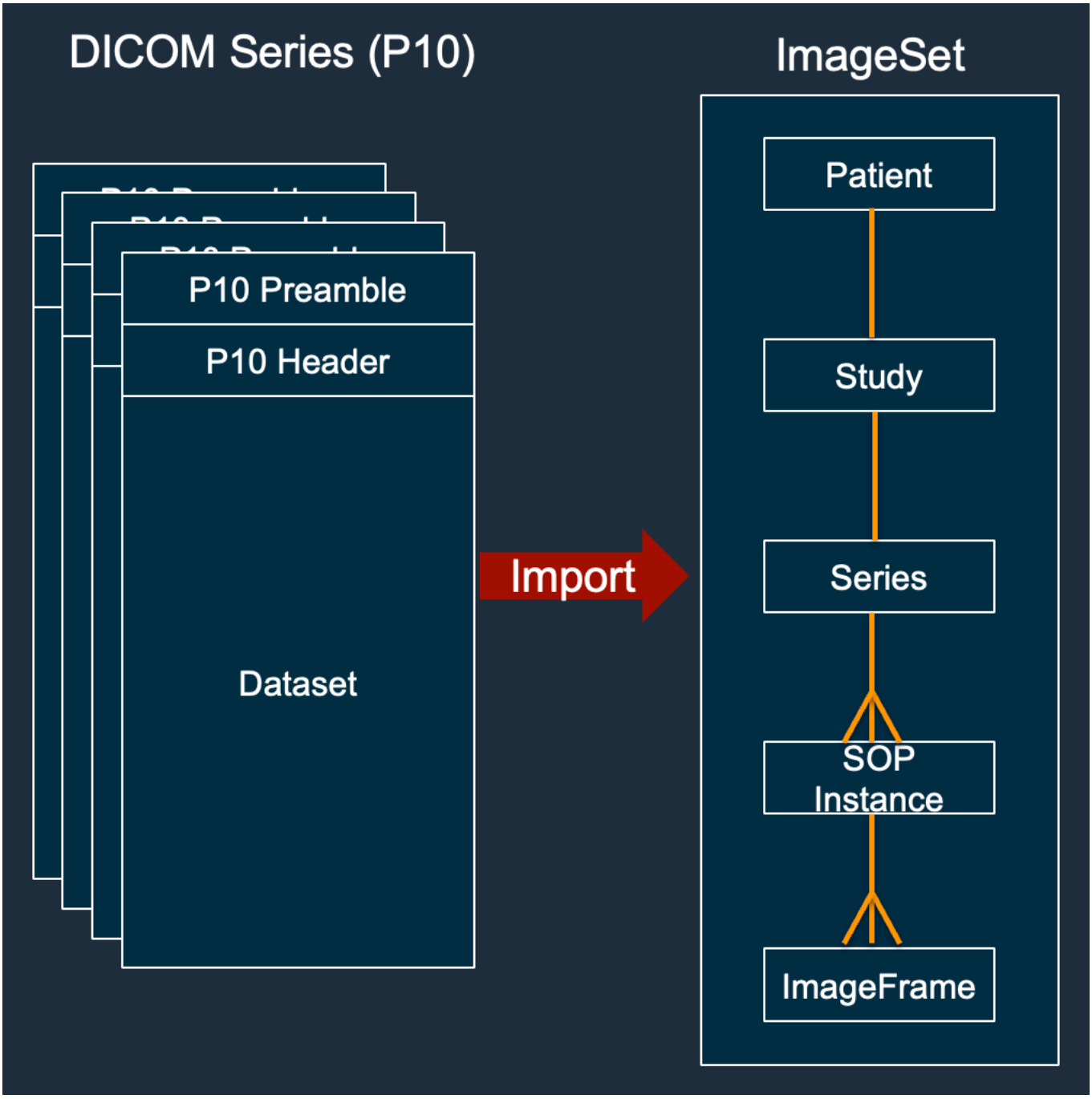
什么是图像集？

影像集是一个 AWS 概念，它定义了一种用于优化相关医学影像数据的抽象分组机制。当您将 DICOM P10 图像数据导入 AWS HealthImaging 数据存储时，它会转换为由元数据和图像框（像素数据）组成的图像集。

> **Note**
>
> 图像集元数据已标准化。换句话说，一组常见的属性和值映射到 DICOM 数据元素注册表中列出的患者、研究和系列级别的元素。
> 影像帧（像素数据）采用高吞吐量 JPEG 2000 (HTJ2K) 编码，必须解码才能观看。

图像集是 AWS 资源，因此它们被分配了 Amazon 资源名称 (ARN)。它们可以用多达 50 个键密钥对进行标记，并通过 IAM 授予基于角色的访问控制 (RBAC)和基于属性的访问权限控制 (ABAC)。此外，还对图像集进行了版本控制，因此所有更改都将保留下来，并且可以访问以前的版本。

导入 DICOM P10 数据会生成包含同一 DICOM 系列中一个或多个服务对象对 (SOP, Service-Object Pair) 实例的 DICOM 元数据和影像帧的影像集。



Note

DICOM 导入任务：

- 始终创建新的图像集，从不更新现有图像集。
- 请勿删除重复的 SOP 实例存储，因为每次导入同一 SOP 实例都会使用额外的存储空间。

- 可以为单个 DICOM 系列创建多个影像集。例如，当标准化元数据属性存在变体时，例如PatientName不匹配。

影像集元数据是什么样子？

您可以使用GetImageSetMetadata操作来检索影像集元数据。返回的元数据是用压缩的gzip，因此在查看之前必须将其解压缩。有关更多信息，请参阅 获取影像集元数据。

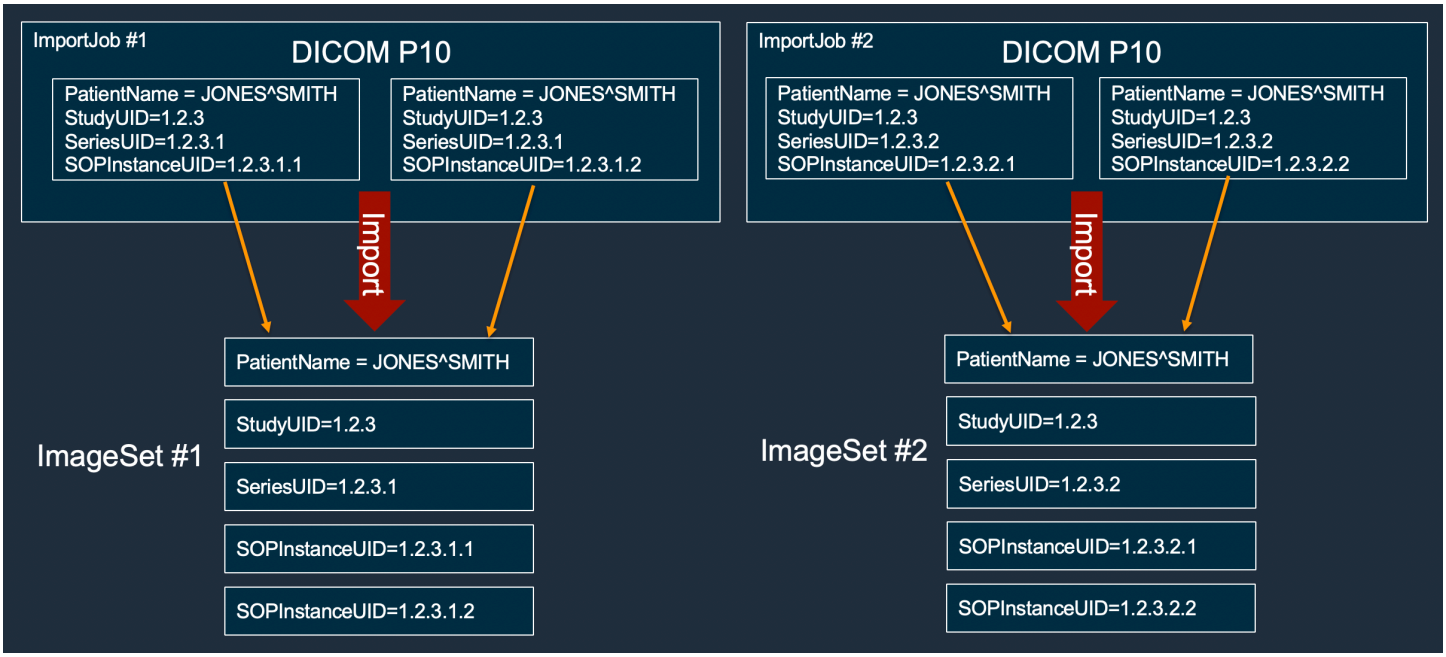
以下示例显示了 JSON 格式的图像集元数据的结构。

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

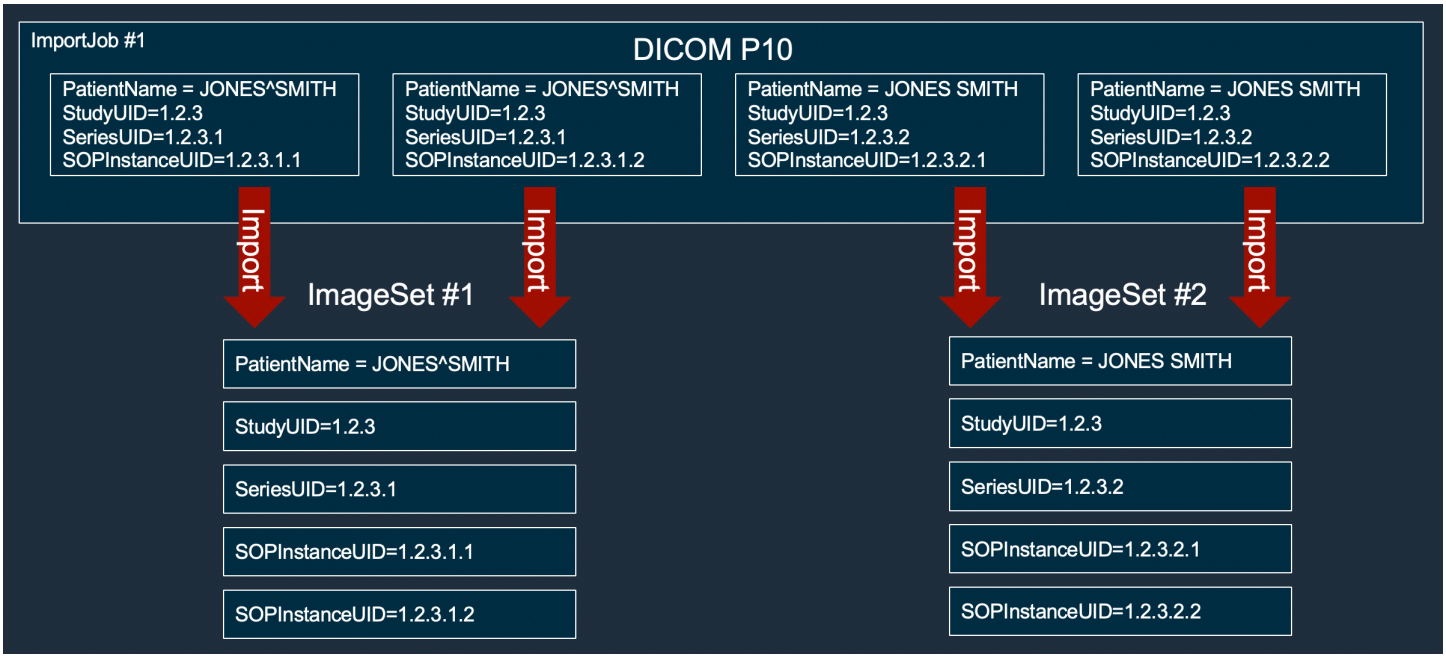
图像集创建示例：多个导入任务

以下示例显示了多个导入任务如何始终创建新的影像集而从不向现有图像集添加图像集。



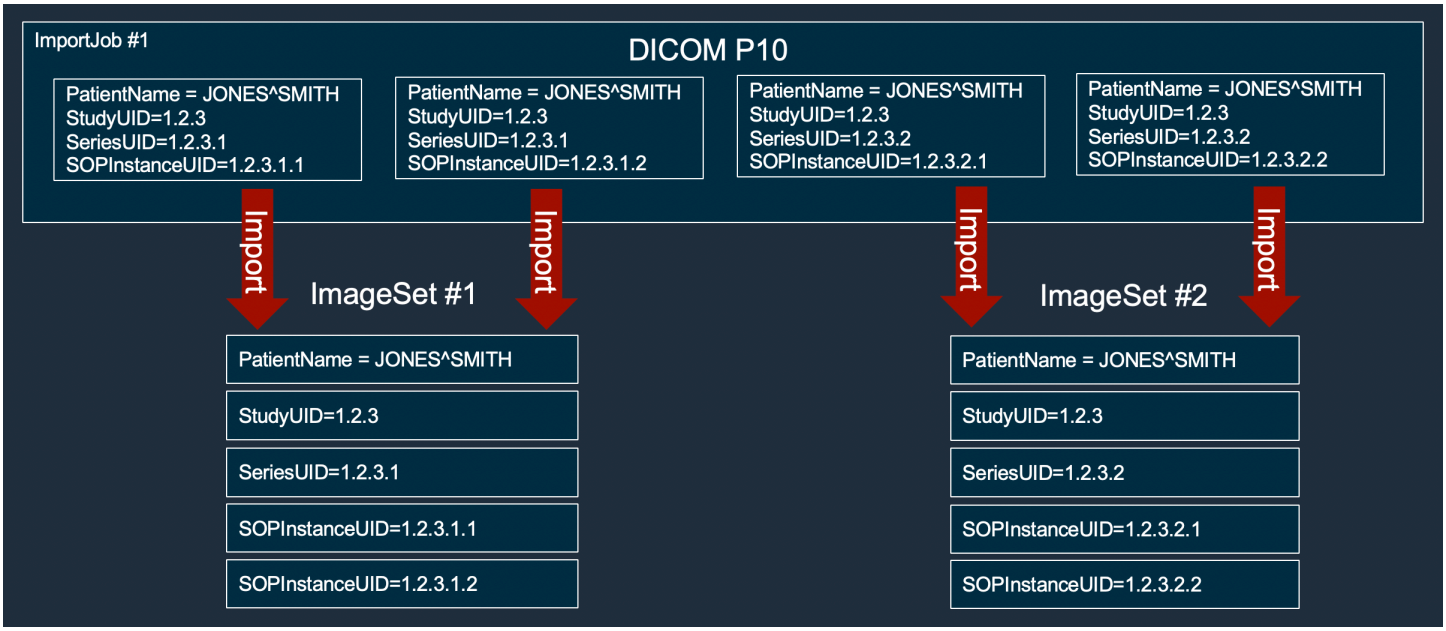
图像集创建示例：具有两个变体的单个导入任务

以下示例显示了创建两个图像集的单个导入任务，因为实例 1 和 2 的患者姓名与实例 3 和 4 的患者姓名不同。



图像集创建示例：经过优化的单个导入任务

以下示例显示了单个导入任务创建了两个图像集以提高吞吐量，即使患者姓名相匹配。



搜索影像集

您可以使用该`SearchImageSets`操作对`ACTIVE` HealthImaging 数据存储中的所有影像集运行搜索查询。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`SearchImageSets`*中的。

> **Note**
>
> 搜索影像集时，请记住以下几点。
>
> - `SearchImageSets` 接受单个搜索查询参数并返回具有匹配条件的所有影像集的分页响应。所有日期范围查询都必须输入为`(lowerBound, upperBound)`.
> - 默认情况下，`SearchImageSets`使用该`updatedAt`字段按从最新到最旧的降序顺序进行排序。
> - 如果您使用客户拥有的密钥创建数据存储，则必须先更新 AWS KMS 密 AWS KMS 钥策略，然后才能与影像集进行交互。更多信息，请参阅创建客户托管式密钥。

搜索影像集

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

> **Note**
>
> 以下过程说明如何使用`Series Instance UID`和`Updated at`属性过滤器搜索影像集。

Series Instance UID

使用`Series Instance UID`属性过滤器搜索影像集

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开，默认情况下，影像集选项卡处于选中状态。
3. 选择属性筛选菜单并选择`Series Instance UID`。
4. 在 “输入要搜索的值” 字段中，输入（粘贴）感兴趣的系列实例 UID。

   > **Note**
   >
   > 系列实例 UID 值必须与 DI COM 唯一标识符 (UID) 注册表中列出的值相同。请注意，要求包括一系列数字，这些数字之间至少包含一个句点。系列实例 UID 的开头或结尾不允许有句点。不允许使用字母和空格，因此在复制和粘贴 UID 时要谨慎行事。
5. 选择日期范围菜单，为系列实例 UID 选择日期范围，然后选择应用。
6. 选择搜索。

   默认情况下，位于所选日期范围内的系列实例 UID 将按最新顺序返回。

Updated at

使用`Updated at`属性过滤器搜索影像集

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开，默认情况下，影像集选项卡处于选中状态。
3. 选择属性筛选菜单并选择`Updated at`。

4. 选择“日期范围”菜单，选择图像集的日期范围，然后选择“应用”。
5. 选择搜索。

   默认情况下，位于所选日期范围内的图像集将按最新顺序返回。

AWS CLI 和 SDK

C++

SDK for C++

用于搜索映像集的实用程序函数。

```
//! 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;
}
```

使用案例 #1：EQUAL 运算符。

```
        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;
            }
        }
```

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

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

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

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

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

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

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

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

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

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

```
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
        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;
            }
        }
```

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
        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;
            }
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考SearchImageSets中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

例 1：使用 EQUAL 运算符搜索影像集

以下 `search-image-sets` 代码示例使用 EQUAL 运算符根据特定值搜索影像集。

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

`search-criteria.json` 的内容

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

输出：

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

示例 2：使用 DICOM 和 DICOM StudyDate 使用 BETWEEN 运算符搜索影像集 StudyTime

以下 `search-image-sets` 代码示例搜索在 1990 年 1 月 1 日 (12:00 AM) 至 2023 年 1 月 1 日 (12:00 AM) 之间生成的 DICOM 研究的影像集。

注意：DICOM StudyTime 是可选的。如果不存在，则上午 12:00（一天的开始）是提供用于筛选的日期的时间值。

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

`search-criteria.json` 的内容

```
{
    "filters": [{
        "values": [{
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "19900101",
                "DICOMStudyTime": "000000"
            }
        },
```

```
        {
            "DICOMStudyDateAndTime": {
                "DICOMStudyDate": "20230101",
                "DICOMStudyTime": "000000"
            }
        }],
        "operator": "BETWEEN"
    }]
}
```

输出：

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

例 3：使用 CreatedAt，通过 BETWEEN 运算符搜索影像集（之前保留了时间研究）

以下`search-image-sets`代码示例搜索在 DICOM Studies 保持在 UTC 时区时间范围 HealthImaging 之间的影像集。

注意：提供的 createdAt 仅作为示例（“1985-04-12T 23:20:50.52 Z”）。

```
aws medical-imaging search-image-sets \
    --datastore-id 12345678901234567890123456789012 \
```

```
    --search-criteria  file://search-criteria.json
```

search-criteria.json 的内容

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

输出：

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

示例 4：在 DICOM SeriesInstance UID 上使用等号运算符搜索图像集，在 updateDat 上使用介于两者之间搜索图像集，然后在 updateDat 字段上按照 ASC 顺序对响应进行排序

以下search-image-sets代码示例在 DICOM SeriesInstance UID 上搜索具有等于运算符的影像集，在 updateDat 上使用介于两者之间的影像集，并在 updateDat 字段上按照 ASC 顺序对响应进行排序。

注意：以示例格式提供 updateDat（“1985-04-12T 23:20:50.52 Z”）。

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

search-criteria.json 的内容

```
{
    "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"
    }
}
```

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的搜索图像集。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SearchImageSets中的。

Java

适用于 Java 2.x 的 SDK

用于搜索映像集的实用程序函数。

```
    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;
    }
```

使用案例 #1：EQUAL 运算符。

```
        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();
        }
```

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

```
        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();
        }
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
        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();
        }
```

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
        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();
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考SearchImageSets中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

用于搜索映像集的实用程序函数。

```
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;
};
```

使用案例 #1：EQUAL 运算符。

```
    const datastoreId = "12345678901234567890123456789012";
```

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

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

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

```
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);
}
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
const datastoreId = "1234567890123456789012345678901 2";

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

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

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
const datastoreId = "1234567890123456789012345678901 2";

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);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考SearchImageSets中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

用于搜索映像集的实用程序函数。

```
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
```

使用案例 #1：EQUAL 运算符。

```
        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}")
```

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

```
        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}"
        )
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
        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}"
        )
```

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
        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}")
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用SearchImageSets于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取影像集属性

您可以使用`GetImageSet`操作返回中给定图像集的属性 HealthImaging。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`GetImageSet`*中的。

> **Note**
>
> 默认情况下，AWS 会 HealthImaging 返回最新版本的图像集的属性。要查看旧版本影像集的属性，请在提交请求时提供 `versionId`。

获取影像集属性

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开，默认情况下，影像集选项卡处于选中状态。
3. 选择一个影像集。

   影像集详细信息页面打开并显示影像集属性。

AWS CLI 和 SDK

CLI

AWS CLI

获取影像集属性

以下 `get-image-set` 代码示例可获取影像集的属性。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的获取图像集属性。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetImageSet中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

```
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用GetImageSet于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取影像集元数据

您可以使用`GetImageSetMetadata`操作来检索中给定图像集的元数据 HealthImaging。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`GetImageSetMetadata`*中的。

> **Note**
>
> 默认情况下， HealthImaging 返回最新版本影像集的元数据属性。要查看旧版本影像集的元数据，请在请求中提供 `versionId`。
> 影像集元数据使用 `gzip` 压缩并以 JSON 对象的形式返回。因此，在查看标准化元数据之前，必须先解压缩 JSON 对象。有关更多信息，请参阅 元数据标准化。

如要获取影像集元数据

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开，默认情况下，影像集选项卡处于选中状态。
3. 选择一个影像集。

   影像集详细信息页面打开，影像集元数据查看器一节下方显示影像集元数据。

AWS CLI 和 SDK

C++

SDK for C++

用于获取映像集元数据的实用程序函数。

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

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

    return outcome.IsSuccess();
}
```

获取没有版本的映像集元数据。

```
        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;
        }
```

获取带有版本的映像集元数据。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考GetImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

例 1：获取没有版本的影像集元数据

以下 `get-image-set-metadata` 代码示例可获取未指定版本的影像集的元数据。

注意：`outfile` 是必需的参数。

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

返回的元数据使用 gzip 压缩并存储在 studymetadata.json.gz 文件中。要查看返回的 JSON 对象的内容，必须先将其解压。

输出：

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

例 2：获取带有版本的影像集元数据

以下 `get-image-set-metadata` 代码示例可获取指定版本的影像集的元数据。

注意：`outfile` 是必需的参数。

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

```
    --version-id 1 \
    studymetadata.json.gz
```

返回的元数据使用 gzip 压缩并存储在 studymetadata.json.gz 文件中。要查看返回的 JSON 对象的内容，必须先将其解压。

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的获取图像集元数据。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetImageSetMetadata中的。

Java

适用于 Java 2.x 的 SDK

```
   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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

用于获取映像集元数据的实用程序函数。

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

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

  if (versionID) {
```

```
    params.versionID = versionID;
  }

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

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

  return response;
};
```

获取没有版本的映像集元数据。

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

获取带有版本的映像集元数据。

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

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考[GetImageSetMetadata](#)中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#)中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

用于获取映像集元数据的实用程序函数。

```
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
```

获取没有版本的映像集元数据。

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

获取带有版本的映像集元数据。

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

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用GetImageSetMetadata于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取影像集像素数据

影像帧是影像集中存在的用于构成 2D 医学影像的像素数据。您可以使用GetImageFrame操作来检索中设置的给定图像的 HTJ2K 编码图像帧。 HealthImaging以下菜单提供了 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*GetImageFrame*中的。

> **Note**
>
> 在导入过程中，AWS 以 HTJ2K 无损格式对所有图像帧进行 HealthImaging 编码，因此，在图像查看器中查看之前，必须对其进行解码。有关更多信息，请参阅 HTJ2K 解码库。

获取影像帧

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

> **Note**
>
> 由于 AWS Management Console未提供影像查看器，因此必须以编程方式对影像帧进行解码和访问。
> 有关解码和查看影像帧的更多信息，请参阅 HTJ2K 解码库。

AWS CLI 和 SDK

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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考GetImageFrame中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

获取影像集像素数据

以下 `get-image-frame` 代码示例可获取影像帧。

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

注意：此代码示例不包括输出，因为该 GetImageFrame 操作将像素数据流返回到 imageframe.jph 文件。有关解码和查看影像帧的信息，请参阅 HTJ2K 解码库。

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的获取图像集像素数据。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetImageFrame中的。

Java

适用于 Java 2.x 的 SDK

```
       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);
                }
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetImageFrame中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetImageFrame中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用GetImageFrame于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取 DICOM 实例

> **Note**
>
> 该 HealthImaging GetDICOMInstance API的构建符合基于网络的医学成像的 DicomWeb检索（WADO-RS）标准。因为GetDICOMInstance是 DicomWeb 服务的代表，所以它不是通过 AWS CLI 和 AWS SDK 提供的。

您可以通过指定GetDICOMInstance与资源关联的系列、研究和实例 UID 从 HealthImaging 数据存储中检索 DICOM 实例。您可以通过提供图像集 ID 作为查询参数来指定应从中检索实例资源的图像集。此外，您可以选择传输语法来压缩 DICOM 数据，并支持未压缩 (ELE) 或高吞吐量 JPEG 2000 (HTJ2K)。借助GetDICOMInstance，您可以与使用 DICOM 第 10 部分二进制文件的系统进行互操作，同时利用 HealthImaging的云原生接口。

获取 DICOM 实例（**.dcm**文件）

1. 收集 HealthImaging datastoreId和imageSetId参数值。
2. 使用带有datastoreId和imageSetId参数值的GetImageSetMetadata操作来检索studyInstanceUIDseriesInstanceUID、和的关联元数据值sopInstanceUID。有关更多信息，请参阅 获取影像集元数据。
3. 使用datastoreId、、、studyInstanceUID和的值为请求构造一个 URL imageSetId。seriesInstanceUID sopInstanceUID网址的格式为：

   ```
   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. 准备并发送您的请求。 GetDICOMInstance使用带有AWS 签名版本 4 签名协议的 HTTP GET 请求。以下代码示例使用curl命令行工具从 HealthImaging中获取 DICOM 实例（.dcm文件）。

   Shell

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

```
instances/1.3.6.1.4.1.5962.1.1.4.1.1.20040826186059.5457?
imageSetId=459e50687f121185f747b67bb60d1bc8' \
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/dicom; transfer-syntax=1.2.840.10008.1.2.1' \
  --output 'dicom-instance.dcm'
```

Note

`transfer-syntax`UID 是可选的，如果不包括在内，则默认为 Explicit VR Little Endian。支持的传输语法包括：

- 显式 VR Little Endian (ELE)-`1.2.840.10008.1.2.1`（默认）
- 带有 RPCL 选项的高吞吐量 JPEG 2000 图像压缩（仅限无损）-`1.2.840.10008.1.2.4.202`

有关更多信息，请参阅 适用于 AWS 的 HTJ2K 解码库 HealthImaging。

使用 AWS HealthImaging 修改图像集

DICOM 导入任务通常要求您修改图像集，原因如下：

- 患者安全
- 数据一致性
- 减少数据存储成本

HealthImaging 提供了多个 API 来简化图像集修改过程。以下主题介绍如何使用 AWS CLI 和 AWS 开发工具包修改图像集。

主题

列出影像集版本

您可以使用`ListImageSetVersions`操作列出中设置的图像的版本历史记录 HealthImaging。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*`ListImageSetVersions`*中的。

> **Note**
>
> AWS 会 HealthImaging 记录对图像集所做的每一次更改。更新影像集元数据会在影像集历史记录中创建新版本。有关更多信息，请参阅 更新影像集元数据。

列出影像集的版本

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。

2. 选择数据存储。

   数据存储详细信息页面将会打开，默认情况下，影像集选项卡处于选中状态。

3. 选择一个影像集。

   将打开影像集详细信息 页面。

   影像集版本显示在影像集详细信息一节下。

AWS CLI 和 SDK

CLI

AWS CLI

列出影像集版本

以下 `list-image-set-versions` 代码示例列出了影像集的版本历史记录。

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

输出：

```
{
    "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
        }
    ]
}
```

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的列出图像集版本。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListImageSetVersions中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListImageSetVersions中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

  const commandParams = { datastoreId, imageSetId };
```

```
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListImageSetVersions中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用ListImageSetVersions于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

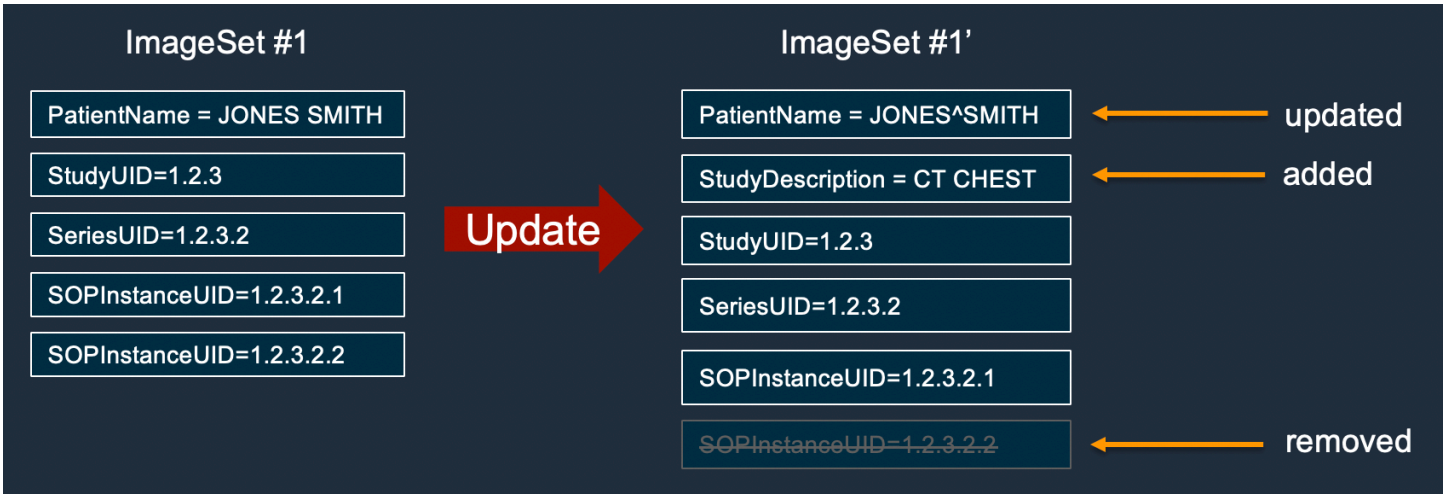
更新影像集元数据

您可以使用UpdateImageSetMetadata操作更新 AWS 中的图像集元数据 HealthImaging。您可以使用此异步过程添加、更新和移除影像集元数据属性，这些属性是导入期间创建的 DICOM 标准化元素的表现形式。使用 UpdateImageSetMetadata 操作，您还可以移除 Series 和 SOP 实例，以使影像集与外部系统保持同步，并取消对影像集元数据的识别。有关更多信息，请参阅 AWS HealthImaging API 参考*UpdateImageSetMetadata*中的。

了解影像集元数据更新

> **Note**
>
> 真实的 DICOM 导入需要更新、添加和移除影像集元数据中的属性。更新影像集元数据时，请记住以下几点：
>
> - 更新影像集元数据会在影像集历史记录中创建新版本。有关更多信息，请参阅 列出影像集版本。
> - 更新映像集元数据是一个异步过程。因此 imageSetState，imageSetWorkflowStatus响应元素可用于提供锁定影像集的相应状态和状态。您无法对锁定的图像集执行其他写入操作。
> - DICOM 元素约束适用于元数据更新。有关更多信息，请参阅 DICOM 元数据限制。
> - 如果图像集元数据更新操作不成功，请调用并查看message响应元素。

下图表示中正在更新的影像集元数据 HealthImaging。



更新影像集元数据

根据您对 AWS 的访问偏好选择一个选项卡 HealthImaging。

AWS CLI 和 SDK

CLI

AWS CLI

在影像集元数据中插入或更新属性

以下update-image-set-metadata代码示例在影像集元数据中插入或更新属性。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "updatableAttributes":
 "eyJTY2hlbWFWZXJzaW9uIjoxLjEsIlBhdGllbnQiOnsiRElDT00iOnsiUGF0aWVudE5hbWUiOiJNWF5NWCJ9fX0
    }
}
```

注意：updatableAttributes 是一个采用 Base64 编码的 JSON 字符串。这是未编码的 JSON 字符串。

{" SchemaVersion ": 1.1，"Patient"：{"DICOM": {" PatientName ": "MX^MX"}}}

输出：

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

从影像集元数据中移除属性

以下update-image-set-metadata代码示例从影像集元数据中移除一个属性。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "removableAttributes":
 "e1NjaGVtYVZlcnNpb246MS4xLFN0dWR5OntESUNPTTp7U3R1ZHlEZXNjcmlwdGlvbjpDSEVTVH19fQo="
    }
}
```

注意：removableAttributes 是一个采用 Base64 编码的 JSON 字符串。这是未编码的 JSON 字符串。键和值必须与要删除的属性相匹配。

{" SchemaVersion ": 1.1，"Study": {"DICOM": {" StudyDescription ": "CHEST"}}}

输出：

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

从影像集元数据中移除实例

以下update-image-set-metadata代码示例从影像集元数据中移除实例。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "removableAttributes":
 "eezEuMS4xLjEuMS4xLjEyMzQ1LjEyMzQ1Njc4OTAxMi4xMjMuMTIzNDU2Nzg5MDEyMzQuMTp7SW5zdGFuY2VzOn
    }
}
```

注意：removableAttributes 是一个采用 Base64 编码的 JSON 字符串。这是未编码的 JSON 字符串。

{“1.1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1”：{“实例”：{“1.1.1.1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1"：{}}}}}

输出：

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

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的更新图像集元数据。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考UpdateImageSetMetadata中的。

Java

适用于 Java 2.x 的 SDK

```
   public static void updateMedicalImageSetMetadata(MedicalImagingClient
medicalImagingClient,
                                                     String datastoreId,
                                                     String imagesetId,
                                                     String versionId,
                                                     MetadataUpdates
metadataUpdates) {
       try {
           UpdateImageSetMetadataRequest updateImageSetMetadataRequest =
UpdateImageSetMetadataRequest
                   .builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId)
                   .latestVersionId(versionId)
                   .updateImageSetMetadataUpdates(metadataUpdates)
                   .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());
           System.exit(1);
       }
   }
```

用例 #1: 插入或更新属性。

```
            final String insertAttributes = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "DICOM": {
                          "StudyDescription": "CT CHEST"
                        }
                      }
                    }
                    """;
            MetadataUpdates metadataInsertUpdates = MetadataUpdates.builder()
                    .dicomUpdates(DICOMUpdates.builder()
                            .updatableAttributes(SdkBytes.fromByteBuffer(
                                    ByteBuffer.wrap(insertAttributes
                                            .getBytes(StandardCharsets.UTF_8))))
                            .build())
                    .build();

            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                    versionid, metadataInsertUpdates);
```

用例 #2: 移除属性。

```
            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);
```

用例 #3: 移除实例。

```
            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);
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考UpdateImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
```

```
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
};
```

用例 #1: 插入或更新属性。

```
        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);
```

用例 #2: 移除属性。

```
        // 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);
```

用例 #3: 移除实例。

```
        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);
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考UpdateImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
    ):
        """
        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\"}}}"}}
        :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,
            )
        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
```

以下代码实例化对象。 MedicalImagingWrapper

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

用例 #1: 插入或更新属性。

```
            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
            )
```

用例 #2: 移除属性。

```
            # 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
            )
```

用例 #3: 移除实例。

```
            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
            )
```

- 有关 API 的详细信息，请参阅适用[UpdateImageSetMetadata](于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库]中查找完整示例，了解如何进行设置和运行。

复制图像集

您可以使用CopyImageSet操作来复制中的图像集 HealthImaging。您可以使用此异步过程将图像集的内容复制到新的或现有的图像集中。您可以复制到新图像中以拆分图像集，也可以创建单独的副本。您还可以复制到现有图像集中，将两个图像集合并在一起。有关更多信息，请参阅 AWS HealthImaging API 参考*CopyImageSet*中的。

了解影像集副本

> **Note**
>
> 复制影像集时，请记住以下几点：
>
> - 复制图像集会在图像集历史记录中创建新版本。有关更多信息，请参阅 列出影像集版本。
> - 复制图像集是一个异步过程。因此，state (`imageSetState`) 和 status (`imageSetWorkflowStatus`) 响应元素可用于让您知道锁定影像集上正在发生什么操作。无法对锁定的影像集执行其他写入操作。
> - CopyImageSet需要唯一的 SOP 实例 UID 才能成功。因此，您必须通过将其从不需要的图片集中移除来选择正确的 SOP 实例。
> - 如果影像集复制操作不成功，请致电 GetImageSet 并查看 `message` 属性。有关更多信息，请参阅 获取影像集属性。
> - 实际导入 DICOM 可能会导致每个 DICOM 系列生成多个图像集。使用 CopyImageSet 操作时，请考虑以下几点：
>   - 将实例从一个图像集复制到另一个图像集
>   - 复制要求两个影像集具有一致的元数据

复制图像集

根据您对 AWS 的访问偏好选择一个选项卡 HealthImaging。

AWS CLI 和 SDK

CLI

AWS CLI

例 1：复制没有目标的影像集。

以下 copy-image-set 代码示例可制作没有目标的影像集的副本。

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

输出：

```
{
    "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"
}
```

例 2：复制带有目标的影像集。

以下 copy-image-set 代码示例可制作带有目标的影像集的副本。

```
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"} }'
```

输出：

```
{
    "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"
}
```

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的复制图像集。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CopyImageSet中的。

Java

适用于 Java 2.x 的 SDK

```
   public static String copyMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imageSetId,
           String latestVersionId,
           String destinationImageSetId,
           String destinationVersionId) {

       try {
           CopySourceImageSetInformation copySourceImageSetInformation =
CopySourceImageSetInformation.builder()
                   .latestVersionId(latestVersionId)
                   .build();
```

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

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

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

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

        return "";
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CopyImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

用于复制映像集的实用程序函数。

```
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.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
```

```
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //      },
  //       datastoreId: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

复制没有目标的映像集。

```
  try {
    await copyImageSet(
      "1234567890123456789012345678901 2",
      "1234567890123456789012345678901 2",
      "1"
    );
  } catch (err) {
```

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

复制带有目标的映像集。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考CopyImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

用于复制映像集的实用程序函数。

```
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,
    ):
        """
        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.
        :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,
                }
            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
        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"]
```

复制没有目标的映像集。

```
            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,
            )
```

复制带有目标的映像集。

```
            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,
            )
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用CopyImageSet于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除一个影像集

您可以使用DeleteImageSet操作删除中设置的图像 HealthImaging。以下菜单提供了操作步骤 AWS Management Console 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。有关更多信息，请参阅 AWS HealthImaging API 参考*DeleteImageSet*中的。

删除映像集

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开，默认情况下，影像集选项卡处于选中状态。
3. 选择影像集并选择删除。

   删除影像集模式打开。
4. 提供影像集的 ID，然后选择删除影像集。

AWS CLI 和 SDK

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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DeleteImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

删除映像集

以下 `delete-image-set` 代码示例可删除影像集。

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

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的删除图像集。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteImageSet中的。

Java

适用于 Java 2.x 的 SDK

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

```
  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用DeleteImageSet于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 AWS 为资源添加标签 HealthImaging

您可以以标签的形式向 AWS HealthImaging 资源（数据存储和图像集）分配元数据。每个标签都是由用户定义的键和值组成的标签。标签帮助您管理、识别、组织、搜索和筛选资源。

> **重要提示**
>
> 请勿在标签中存储受保护的健康信息（PHI）、个人身份信息（PII）或其他机密或敏感信息。标签不适合用于私有或敏感数据。

以下主题介绍如何使用 AWS Management Console AWS CLI、和 AWS 软件开发工具包使用 HealthImaging 标记操作。有关更多信息，请参阅AWS 一般参考 指南中的为AWS 资源添加标签。

主题

标记资源

要在 AWS 中为资源添加标签 HealthImaging，请使用`TagResource`操作。以下代码示例描述了如何将`TagResource`操作与 AWS Management Console AWS CLI、和 AWS SDK 配合使用。有关更多信息，请参阅AWS 一般参考 指南中的为AWS 资源添加标签。

标记资源（数据存储）

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开。
3. 选择详细信息选项卡。

4. 在 标签 部分中，选择 管理标签。

   将打开管理标签页面。

5. 选择 添加新标签。
6. 输入一个 键和可选的 值（可选）。
7. 选择 保存更改。

AWS CLI 和 SDK

CLI

AWS CLI

例 1：标记数据存储

以下 `tag-resource` 代码示例可标记数据存储。

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

此命令不生成任何输出。

例 2：标记影像集

以下 `tag-resource` 代码示例可标记影像集。

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

此命令不生成任何输出。

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》 AWS HealthImaging中的使用为资源添加标签。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考TagResource中的。

Java

适用于 Java 2.x 的 SDK

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考TagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考TagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用TagResource于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出资源的标签

要在 AWS 中列出资源的标签 HealthImaging，您可以使用ListTagsForResource操作。以下代码示例描述了如何将ListTagsForResource操作与 AWS Management Console AWS CLI、和 AWS SDK 配合使用。有关更多信息，请参阅AWS 一般参考 指南中的为AWS 资源添加标签。

列出资源（数据存储）的标签

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开。
3. 选择详细信息选项卡。

   在标签部分下，列出了所有数据存储标签。

AWS CLI 和 SDK

CLI

AWS CLI

例 1：列出数据存储的资源标签

以下 `list-tags-for-resource` 代码示例列出数据存储的标签。

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

输出：

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

例 2：列出影像集的资源标签

以下 `list-tags-for-resource` 代码示例列出影像集的标签。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》 AWS HealthImaging中的使用为资源添加标签。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListTagsForResource中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListTagsForResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListTagsForResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用ListTagsForResource于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

取消标记资源

要在 AWS 中取消对资源的标记 HealthImaging，您可以使用`UntagResource`操作。以下代码示例描述了如何将`UntagResource`操作与 AWS Management Console AWS CLI、和 AWS SDK 配合使用。有关更多信息，请参阅AWS 一般参考 指南中的为AWS 资源添加标签。

取消标记资源（数据存储）

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

AWS 控制台

1. 打开 HealthImaging 控制台数据存储页面。
2. 选择数据存储。

   数据存储详细信息页面将会打开。
3. 选择详细信息选项卡。
4. 在 标签 部分中，选择 管理标签。

   将打开管理标签页面。
5. 在标签旁选择 移除，以移除标签。
6. 选择 保存更改。

AWS CLI 和 SDK

CLI

AWS CLI

例 1：取消标记数据存储

以下 `untag-resource` 代码示例可取消标记数据存储。

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

此命令不生成任何输出。

例 2：取消标记影像集

以下 `untag-resource` 代码示例可取消标记影像集。

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

此命令不生成任何输出。

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》 AWS HealthImaging中的使用为资源添加标签。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考UntagResource中的。

Java

适用于 Java 2.x 的 SDK

```
    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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考UntagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考UntagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用UntagResource于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

HealthImaging 使用 AWS SDK 的代码示例

以下代码示例说明如何 HealthImaging 使用 AWS 软件开发套件 (SDK)。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

你好 HealthImaging

以下代码示例展示了如何开始使用 HealthImaging。

C++

SDK for C++

C MakeLists .txt cMake 文件的代码。

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

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

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

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

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

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
```

```
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

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

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

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

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

add_executable(${PROJECT_NAME}
        hello_health_imaging.cpp)

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

hello_health_imaging.cpp 源文件代码。

```
#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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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 Amazon HealthImaging
    client and list the data stores in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param medical_imaging_client: A Boto3 Amazon HealthImaging Client object.
    """
    print("Hello, Amazon Health Imaging! Let's list some of your data stores:\n")
```

```
    try:
        paginator = medical_imaging_client.get_paginator("list_datastores")
        page_iterator = paginator.paginate()
        datastore_summaries = []
        for page in page_iterator:
            datastore_summaries.extend(page["datastoreSummaries"])
        print("\tData Stores:")
        for ds in datastore_summaries:
            print(f"\t\tDatastore: {ds['datastoreName']} ID {ds['datastoreId']}")
    except ClientError as err:
        logger.error(
            "Couldn't list data stores. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise


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

- 有关 API 的详细信息，请参阅适用ListDatastores于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

代码示例

- HealthImaging 使用 AWS SDK 的操作
  - CopyImageSet与 AWS SDK 或 CLI 配合使用
  - CreateDatastore与 AWS SDK 或 CLI 配合使用
  - DeleteDatastore与 AWS SDK 或 CLI 配合使用
  - DeleteImageSet与 AWS SDK 或 CLI 配合使用
  - GetDICOMImportJob与 AWS SDK 或 CLI 配合使用
  - GetDatastore与 AWS SDK 或 CLI 配合使用
  - GetImageFrame与 AWS SDK 或 CLI 配合使用

- GetImageSet与 AWS SDK 或 CLI 配合使用
- GetImageSetMetadata与 AWS SDK 或 CLI 配合使用
- ListDICOMImportJobs与 AWS SDK 或 CLI 配合使用
- ListDatastores与 AWS SDK 或 CLI 配合使用
- ListImageSetVersions与 AWS SDK 或 CLI 配合使用
- ListTagsForResource与 AWS SDK 或 CLI 配合使用
- SearchImageSets与 AWS SDK 或 CLI 配合使用
- StartDICOMImportJob与 AWS SDK 或 CLI 配合使用
- TagResource与 AWS SDK 或 CLI 配合使用
- UntagResource与 AWS SDK 或 CLI 配合使用
- UpdateImageSetMetadata与 AWS SDK 或 CLI 配合使用

- HealthImaging 使用 AWS SDK 的场景
  - 使用 AWS SDK 开始使用 HealthImaging 图像集和图像框架
  - 使用 S HealthImaging DK 为数据存储添加标签 AWS
  - 使用 SDK 为 HealthImaging 图像集添加标签 AWS

HealthImaging 使用 AWS SDK 的操作

以下代码示例演示如何使用 AWS SDK 执行单个 HealthImaging操作。这些摘录调用 HealthImaging API，是必须在上下文中运行的大型程序的代码摘录。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 AWS HealthImaging API 参考。

示例

- CopyImageSet与 AWS SDK 或 CLI 配合使用
- CreateDatastore与 AWS SDK 或 CLI 配合使用
- DeleteDatastore与 AWS SDK 或 CLI 配合使用
- DeleteImageSet与 AWS SDK 或 CLI 配合使用
- GetDICOMImportJob与 AWS SDK 或 CLI 配合使用
- GetDatastore与 AWS SDK 或 CLI 配合使用
- GetImageFrame与 AWS SDK 或 CLI 配合使用

- GetImageSet与 AWS SDK 或 CLI 配合使用
- GetImageSetMetadata与 AWS SDK 或 CLI 配合使用
- ListDICOMImportJobs与 AWS SDK 或 CLI 配合使用
- ListDatastores与 AWS SDK 或 CLI 配合使用
- ListImageSetVersions与 AWS SDK 或 CLI 配合使用
- ListTagsForResource与 AWS SDK 或 CLI 配合使用
- SearchImageSets与 AWS SDK 或 CLI 配合使用
- StartDICOMImportJob与 AWS SDK 或 CLI 配合使用
- TagResource与 AWS SDK 或 CLI 配合使用
- UntagResource与 AWS SDK 或 CLI 配合使用
- UpdateImageSetMetadata与 AWS SDK 或 CLI 配合使用

CopyImageSet与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 CopyImageSet。

CLI

AWS CLI

例 1：复制没有目标的影像集。

以下 copy-image-set 代码示例可制作没有目标的影像集的副本。

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

输出：

```
{
    "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"
}
```

例 2：复制带有目标的影像集。

以下 copy-image-set 代码示例可制作带有目标的影像集的副本。

```
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"} }'
```

输出：

```
{
    "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"
}
```

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的复制图像集。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CopyImageSet中的。

Java

适用于 Java 2.x 的 SDK

```
   public static String copyMedicalImageSet(MedicalImagingClient
medicalImagingClient,
            String datastoreId,
            String imageSetId,
            String latestVersionId,
            String destinationImageSetId,
            String destinationVersionId) {

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

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

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

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

```
            CopyImageSetResponse response =
 medicalImagingClient.copyImageSet(copyImageSetRequest);

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

        return "";
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CopyImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

用于复制映像集的实用程序函数。

```
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.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
```

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

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //      },
  //       datastoreId: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
```

```
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

复制没有目标的映像集。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

复制带有目标的映像集。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考CopyImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

用于复制映像集的实用程序函数。

```
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,
    ):
        """
        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.
        :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,
                }
            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
        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"]
```

复制没有目标的映像集。

```
            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,
            )
```

复制带有目标的映像集。

```
            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,
        )
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用CopyImageSet于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`CreateDatastore`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateDatastore`。

Bash

AWS CLI 使用 Bash 脚本

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

```
########################################################################
# function imaging_create_datastore
#
# This function creates an AWS HealthImaging data store for importing DICOM P10
 files.
#
# Parameters:
#       -n data_store_name - The name of the data store.
#
# Returns:
#       The datastore ID.
#    And:
#       0 - If successful.
#       1 - If it fails.
########################################################################
function imaging_create_datastore() {
  local datastore_name response
  local option OPTARG # Required to use getopts command in a function.

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

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

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

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

  local error_code=${?}

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

  echo "$response"

  return 0
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

创建数据存储

以下 `create-datastore` 代码示例创建名称为 `my-datastore` 的数据存储。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的创建数据存储。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateDatastore中的。

Java

适用于 Java 2.x 的 SDK

```
   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 "";
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考CreateDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用CreateDatastore于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteDatastore`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteDatastore`。

Bash

AWS CLI 使用 Bash 脚本

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

##############################################################################
# function imaging_delete_datastore
#
# This function deletes an AWS HealthImaging data store.
#
# Parameters:
#       -i datastore_id - The ID of the data store.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
```

```
##############################################################################
function imaging_delete_datastore() {
  local datastore_id response
  local option OPTARG # Required to use getopts command in a function.

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

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

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

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

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports medical-imaging delete-datastore operation
 failed.$response"
    return 1
  fi

  return 0
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

删除数据存储

以下 `delete-datastore` 代码示例可删除数据存储。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的删除数据存储。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteDatastore中的。

Java

适用于 Java 2.x 的 SDK

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //         },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用DeleteDatastore于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteImageSet`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteImageSet`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用影像集和影像帧

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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DeleteImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

删除映像集

以下 delete-image-set 代码示例可删除影像集。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的删除图像集。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteImageSet中的。

Java

适用于 Java 2.x 的 SDK

```
   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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

```
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用DeleteImageSet于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetDICOMImportJob`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetDICOMImportJob`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用影像集和影像帧

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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考ImportJob中的 getDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

获取 DICOM 导入任务的属性

以下 get-dicom-import-job 代码示例可获取导入任务的属性。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的获取导入任务属性。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》ImportJob中的 getDiCom。

Java

适用于 Java 2.x 的 SDK

```
    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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x AP I 参考ImportJob中的 getDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

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

  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ImportJob中的 getDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅 Pyt hon AWS SDK (Boto3) API 参考ImportJob中的 getDiCom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetDatastore`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetDatastore`。

Bash

AWS CLI 使用 Bash 脚本

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

##############################################################################
# function imaging_get_datastore
#
# Get a data store's properties.
#
# Parameters:
#       -i data_store_id - The ID of the data store.
#
# Returns:
#       [datastore_name, datastore_id, datastore_status, datastore_arn,
 created_at, updated_at]
#    And:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function imaging_get_datastore() {
```

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

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

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

 local response

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

```
  )
  error_code=${?}

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

  echo "$response"

  return 0
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

获取数据存储的属性

以下 `get-datastore` 代码示例可获取数据存储的属性。

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

输出：

```
{
    "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"
    }
}
```

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的获取数据存储属性。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetDatastore中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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"];
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

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

- 有关 API 的详细信息，请参阅适用GetDatastore于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetImageFrame`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetImageFrame`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用影像集和影像帧

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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考GetImageFrame中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

获取影像集像素数据

以下 get-image-frame 代码示例可获取影像帧。

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

注意：此代码示例不包括输出，因为该 GetImageFrame 操作将像素数据流返回到 imageframe.jph 文件。有关解码和查看影像帧的信息，请参阅 HTJ2K 解码库。

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的获取图像集像素数据。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetImageFrame中的。

Java

适用于 Java 2.x 的 SDK

```
       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);
                }
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetImageFrame中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetImageFrame中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用GetImageFrame于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetImageSet`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetImageSet`。

CLI

AWS CLI

获取影像集属性

以下 `get-image-set` 代码示例可获取影像集的属性。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的获取图像集属性。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetImageSet中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

```
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用GetImageSet于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetImageSetMetadata`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetImageSetMetadata`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用影像集和影像帧

C++

SDK for C++

用于获取映像集元数据的实用程序函数。

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

    return outcome.IsSuccess();
```

```
}
```

获取没有版本的映像集元数据。

```
        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;
        }
```

获取带有版本的映像集元数据。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考GetImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

例 1：获取没有版本的影像集元数据

以下 `get-image-set-metadata` 代码示例可获取未指定版本的影像集的元数据。

注意：`outfile` 是必需的参数。

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

返回的元数据使用 gzip 压缩并存储在 studymetadata.json.gz 文件中。要查看返回的 JSON 对象的内容，必须先将其解压。

输出：

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

例 2：获取带有版本的影像集元数据

以下 `get-image-set-metadata` 代码示例可获取指定版本的影像集的元数据。

注意：`outfile` 是必需的参数。

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

返回的元数据使用 gzip 压缩并存储在 studymetadata.json.gz 文件中。要查看返回的 JSON 对象的内容，必须先将其解压。

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的获取图像集元数据。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetImageSetMetadata中的。

Java

适用于 Java 2.x 的 SDK

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

用于获取映像集元数据的实用程序函数。

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

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

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

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

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

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

  return response;
};
```

获取没有版本的映像集元数据。

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

获取带有版本的映像集元数据。

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

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

用于获取映像集元数据的实用程序函数。

```
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
```

获取没有版本的映像集元数据。

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

获取带有版本的映像集元数据。

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

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用GetImageSetMetadata于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListDICOMImportJobs`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListDICOMImportJobs`。

CLI

AWS CLI

列出 DICOM 导入任务

以下 `list-dicom-import-jobs` 代码示例列出 DICOM 导入任务。

```
aws medical-imaging list-dicom-import-jobs \
    --datastore-id "1234567890123456789012345678901 2"
```

输出：

```
{
    "jobSummaries": [
        {
            "jobId": "09876543210987654321098765432109",
            "jobName": "my-job",
            "jobStatus": "COMPLETED",
            "datastoreId": "1234567890123456789012345678901 2",
            "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"
```

```
        }
    ]
}
```

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的列出导入任务。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》ImportJobs中的 ListDicom。

Java

适用于 Java 2.x 的 SDK

```
   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<>();
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x AP I 参考ImportJobs中的 ListDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

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

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

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

  return jobSummaries;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ImportJobs中的 ListDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅 Python AWS SDK (Boto3) API 参考ImportJobs中的 ListDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListDatastores`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListDatastores`。

Bash

AWS CLI 使用 Bash 脚本

```
###############################################################################
# function errecho
```

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

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

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

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

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

  echo "$response"

  return 0
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

列出数据存储

以下 `list-datastores` 代码示例列出可用的数据存储。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的列出数据存储。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListDatastores中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

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

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
```

```
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用ListDatastores于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListImageSetVersions`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListImageSetVersions`。

CLI

AWS CLI

列出影像集版本

以下 `list-image-set-versions` 代码示例列出了影像集的版本历史记录。

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

输出：

```
{
    "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
        }
    ]
}
```

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的列出图像集版本。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListImageSetVersions中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListImageSetVersions中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

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

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

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

```
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListImageSetVersions中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用ListImageSetVersions于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListTagsForResource`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListTagsForResource`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 标记数据存储
- 标记映像集

CLI

AWS CLI

例 1：列出数据存储的资源标签

以下 `list-tags-for-resource` 代码示例列出数据存储的标签。

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

输出：

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

例 2：列出影像集的资源标签

以下 `list-tags-for-resource` 代码示例列出影像集的标签。

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

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》 AWS HealthImaging中的使用为资源添加标签。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListTagsForResource中的。

Java

适用于 Java 2.x 的 SDK

```
   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;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListTagsForResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListTagsForResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用ListTagsForResource于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`SearchImageSets`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `SearchImageSets`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用影像集和影像帧

C++

SDK for C++

用于搜索映像集的实用程序函数。

```
//! 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;
}
```

使用案例 #1：EQUAL 运算符。

```
        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;
            }
        }
```

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

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

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

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

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

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

```
        .WithDICOMStudyTime("000000.000"));

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

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

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

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

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
        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;
           }
       }
```

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
       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;
            }
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考SearchImageSets中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

例 1：使用 EQUAL 运算符搜索影像集

以下 `search-image-sets` 代码示例使用 EQUAL 运算符根据特定值搜索影像集。

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

search-criteria.json 的内容

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

输出：

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

示例 2：使用 DICOM 和 DICOM StudyDate 使用 BETWEEN 运算符搜索影像集 StudyTime

以下 search-image-sets 代码示例搜索在 1990 年 1 月 1 日 (12:00 AM) 至 2023 年 1 月 1 日 (12:00 AM) 之间生成的 DICOM 研究的影像集。

注意：DICOM StudyTime 是可选的。如果不存在，则上午 12:00（一天的开始）是提供用于筛选的日期的时间值。

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

search-criteria.json 的内容

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

输出：

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

例 3：使用 CreatedAt，通过 BETWEEN 运算符搜索影像集（之前保留了时间研究）

以下`search-image-sets`代码示例搜索在 DICOM Studies 保持在 UTC 时区时间范围 HealthImaging 之间的影像集。

注意：提供的 createdAt 仅作为示例（“1985-04-12T 23:20:50.52 Z”）。

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

`search-criteria.json` 的内容

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

输出：

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

示例 4：在 DICOM SeriesInstance UID 上使用等号运算符搜索图像集，在 updateDat 上使用介于两者之间搜索图像集，然后在 updateDat 字段上按照 ASC 顺序对响应进行排序

以下`search-image-sets`代码示例在 DICOM SeriesInstance UID 上搜索具有等于运算符的影像集，在 updateDat 上使用介于两者之间的影像集，并在 updateDat 字段上按照 ASC 顺序对响应进行排序。

注意：以示例格式提供 updateDat (“1985-04-12T 23:20:50.52 Z”) 。

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

`search-criteria.json` 的内容

```
{
    "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"
    }
}
```

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的搜索图像集。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SearchImageSets中的。

Java

适用于 Java 2.x 的 SDK

用于搜索映像集的实用程序函数。

```
   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;
   }
```

使用案例 #1：EQUAL 运算符。

```
       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();
        }
```

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

```
        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();
        }
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
        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();
        }
```

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
        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();
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考SearchImageSets中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

用于搜索映像集的实用程序函数。

```
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;
};
```

使用案例 #1：EQUAL 运算符。

```
    const datastoreId = "12345678901234567890123456789012";

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

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

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

```
    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);
    }
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
    const datastoreId = "1234567890123456789012345678901 2";

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

```
        };

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

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
    const datastoreId = "1234567890123456789012345678901 2";

    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);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考SearchImageSets中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

用于搜索映像集的实用程序函数。

```
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
```

使用案例 #1：EQUAL 运算符。

```
        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}")
```

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

```
        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}"
        )
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
        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}"
        )
```

用例 #4: DICOM SeriesInstance UID 上的 EQUAL 运算符和 updateDat 上的 BETWEEN 运算符，在 updateDat 字段上按照 ASC 顺序对响应进行排序。

```
        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}")
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用SearchImageSets于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`StartDICOMImportJob`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `StartDICOMImportJob`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用影像集和影像帧

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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ AP I 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CLI

AWS CLI

启动 DICOM 导入任务

以下 `start-dicom-import-job` 代码示例启动 DICOM 导入任务。

```
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"
```

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》中的启动导入任务。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》ImportJob中的 StartDicom。

Java

适用于 Java 2.x 的 SDK

```
    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 "";
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x AP I 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

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

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input
 files.
```

```
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files
 are stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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"]
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅 Python AWS SDK (Boto3) API 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`TagResource`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `TagResource`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 标记数据存储
- 标记映像集

CLI

AWS CLI

例 1：标记数据存储

以下 tag-resource 代码示例可标记数据存储。

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

此命令不生成任何输出。

例 2：标记影像集

以下 tag-resource 代码示例可标记影像集。

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

此命令不生成任何输出。

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》 AWS HealthImaging中的使用为资源添加标签。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考TagResource中的。

Java

适用于 Java 2.x 的 SDK

```
    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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考TagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考TagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用TagResource于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`UntagResource`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `UntagResource`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 标记数据存储
- 标记映像集

CLI

AWS CLI

例 1：取消标记数据存储

以下 untag-resource 代码示例可取消标记数据存储。

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

此命令不生成任何输出。

例 2：取消标记影像集

以下 untag-resource 代码示例可取消标记影像集。

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

此命令不生成任何输出。

有关更多信息，请参阅《AWS HealthImaging 开发人员指南》 AWS HealthImaging中的使用为资源添加标签。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考UntagResource中的。

Java

适用于 Java 2.x 的 SDK

```
    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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[UntagResource](#)中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考UntagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 的详细信息，请参阅适用UntagResource于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

UpdateImageSetMetadata与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 UpdateImageSetMetadata。

CLI

AWS CLI

在影像集元数据中插入或更新属性

以下update-image-set-metadata代码示例在影像集元数据中插入或更新属性。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "updatableAttributes":
 "eyJTY2hlbWFWZXJzaW9uIjoxLjEsIlBhdGllbnQiOnsiRElDT00iOnsiUGF0aWVudE5hbWUiOiJNWF5NWCJ9fX0
    }
}
```

注意：updatableAttributes 是一个采用 Base64 编码的 JSON 字符串。这是未编码的 JSON 字符串。

{” SchemaVersion “: 1.1 , “Patient” : {“DICOM”: {” PatientName “: “MX^MX”}}}

输出：

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

从影像集元数据中移除属性

以下update-image-set-metadata代码示例从影像集元数据中移除一个属性。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
```

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

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "removableAttributes":
 "e1NjaGVtYVZlcnNpb246MS4xLFN0dWR5OntESUNPTTp7U3R1ZHlEZXNjcmlwdGlvbjpDSEVTVH19fQo="
    }
}
```

注意：removableAttributes 是一个采用 Base64 编码的 JSON 字符串。这是未编码的 JSON 字符串。键和值必须与要删除的属性相匹配。

{” SchemaVersion “: 1.1，“Study”: {“DICOM”: {” StudyDescription “: “CHEST”}}}

输出：

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

从影像集元数据中移除实例

以下update-image-set-metadata代码示例从影像集元数据中移除实例。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --latest-version-id 1 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的内容

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

```
        "removableAttributes":
 "eezEuMS4xLjEuMS4xLjEyMzQ1LjEyMzQ1Njc4OTAxMi4xMjMuMTIzNDU2Nzg5MDEyMzQuMTp7SW5zdGFuY2VzOn
    }
}
```

注意：removableAttributes 是一个采用 Base64 编码的 JSON 字符串。这是未编码的 JSON 字符串。

{“1.1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1”：{“实例”：{“1.1.1.1.1.1.1.1.1.1.1.12345.123456789012.123.12345678901234.1"：{}}}}}

输出：

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

有关更多信息，请参阅《AWS HealthImaging 开发者指南》中的更新图像集元数据。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考UpdateImageSetMetadata中的。

Java

适用于 Java 2.x 的 SDK

```
   public static void updateMedicalImageSetMetadata(MedicalImagingClient
medicalImagingClient,
                                                     String datastoreId,
                                                     String imagesetId,
                                                     String versionId,
                                                     MetadataUpdates
metadataUpdates) {
       try {
           UpdateImageSetMetadataRequest updateImageSetMetadataRequest =
UpdateImageSetMetadataRequest
                   .builder()
```

```
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId)
                    .latestVersionId(versionId)
                    .updateImageSetMetadataUpdates(metadataUpdates)
                    .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());
            System.exit(1);
        }
    }
```

用例 #1: 插入或更新属性。

```
            final String insertAttributes = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "DICOM": {
                          "StudyDescription": "CT CHEST"
                        }
                      }
                    }
                    """;
            MetadataUpdates metadataInsertUpdates = MetadataUpdates.builder()
                    .dicomUpdates(DICOMUpdates.builder()
                            .updatableAttributes(SdkBytes.fromByteBuffer(
                                    ByteBuffer.wrap(insertAttributes
                                            .getBytes(StandardCharsets.UTF_8))))
                            .build())
                    .build();

            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                    versionid, metadataInsertUpdates);
```

用例 #2: 移除属性。

```
            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);
```

用例 #3: 移除实例。

```
            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);
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[UpdateImageSetMetadata](https://docs.aws.amazon.com/goto/SdkForJavaV2/medical-imaging-2023-07-19/UpdateImageSetMetadata)中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

```
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.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                              imageSetId = "xxxxxxxxxx",
                                              latestVersionId = "1",
                                              updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
```

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

用例 #1: 插入或更新属性。

```
        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);
```

用例 #2: 移除属性。

```
        // 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);
```

用例 #3: 移除实例。

```
        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);
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考[UpdateImageSetMetadata](#)中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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
    ):
        """
        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\"}}}"}}
        :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,
            )
        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
```

以下代码实例化对象。 MedicalImagingWrapper

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

用例 #1: 插入或更新属性。

```
            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
            )
```

用例 #2: 移除属性。

```
            # 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
            )
```

用例 #3: 移除实例。

```
            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
            )
```

- 有关 API 的详细信息，请参阅适用UpdateImageSetMetadata于 Python 的AWS SDK (Boto3) API 参考。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

HealthImaging 使用 AWS SDK 的场景

以下代码示例向您展示了如何 HealthImaging使用 AWS 软件开发工具包实现常见场景。这些场景向您展示了如何通过在其中调用多个函数来完成特定任务 HealthImaging。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

示例

- 使用 AWS SDK 开始使用 HealthImaging 图像集和图像框架
- 使用 S HealthImaging DK 为数据存储添加标签 AWS
- 使用 SDK 为 HealthImaging 图像集添加标签 AWS

使用 AWS SDK 开始使用 HealthImaging 图像集和图像框架

以下代码示例演示如何导入 DICOM 文件和在中下载图像框架。 HealthImaging

该实现结构为工作流命令行应用程序。

- 设置 DICOM 导入的资源。
- 将 DICOM 文件导入数据存储中。
- 检索导入任务的影像集 ID。
- 检索影像集的影像帧 ID。

- 下载、解码并验证影像帧。
- 清理资源。

C++

SDK for C++

使用必要的资源创建 AWS CloudFormation 堆栈。

```
   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: ");
    }
```

将 DICOM 文件复制到 Amazon S3 导入桶。

```
    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;
    }
```

将 DICOM 文件导入 Amazon S3 数据存储。

```
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;
}
```

获取由 DICOM 导入任务创建的影像集。

```
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;
}
```

获取影像集的影像帧信息。

```
bool AwsDoc::Medical_Imaging::getImageFramesForImageSet(const Aws::String
 &dataStoreID,
                                                        const Aws::String
 &imageSetID,
                                                        const Aws::String
 &outDirectory,

 Aws::Vector<ImageFrameInfo> &imageFrames,
```

```
                                                        const
Aws::Client::ClientConfiguration &clientConfiguration) {
   Aws::String fileName = outDirectory + "/" + imageSetID +
"_metadata.json.gzip";
   bool result = false;
   if (getImageSetMetadata(dataStoreID, imageSetID, "", // Empty string for
version ID.
                           fileName, clientConfiguration)) {
       try {
           std::string metadataGZip;
           {
               std::ifstream inFileStream(fileName.c_str(), std::ios::binary);
               if (!inFileStream) {
                   throw std::runtime_error("Failed to open file " + fileName);
               }

               std::stringstream stringStream;
               stringStream << inFileStream.rdbuf();
               metadataGZip = stringStream.str();
           }
           std::string metadataJson = gzip::decompress(metadataGZip.data(),
                                                       metadataGZip.size());
           // Use JMESPath to extract the image set IDs.
           // https://jmespath.org/specification.html
           jsoncons::json doc = jsoncons::json::parse(metadataJson);
           std::string jmesPathExpression = "Study.Series.*.Instances[].*[]";
           jsoncons::json instances = jsoncons::jmespath::search(doc,

jmesPathExpression);
           for (auto &instance: instances.array_range()) {
               jmesPathExpression = "DICOM.RescaleSlope";
               std::string rescaleSlope = jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();
               jmesPathExpression = "DICOM.RescaleIntercept";
               std::string rescaleIntercept =
jsoncons::jmespath::search(instance,

jmesPathExpression).to_string();

               jmesPathExpression = "ImageFrames[][]";
               jsoncons::json imageFramesJson =
jsoncons::jmespath::search(instance,
```

```
 jmesPathExpression);

                for (auto &imageFrame: imageFramesJson.array_range()) {
                    ImageFrameInfo imageFrameIDs;
                    imageFrameIDs.mImageSetId = imageSetID;
                    imageFrameIDs.mImageFrameId = imageFrame.find(
                            "ID")->value().as_string();
                    imageFrameIDs.mRescaleIntercept = rescaleIntercept;
                    imageFrameIDs.mRescaleSlope = rescaleSlope;
                    imageFrameIDs.MinPixelValue = imageFrame.find(
                            "MinPixelValue")->value().as_string();
                    imageFrameIDs.MaxPixelValue = imageFrame.find(
                            "MaxPixelValue")->value().as_string();

                    jmesPathExpression =
 "max_by(PixelDataChecksumFromBaseToFullResolution, &Width).Checksum";
                    jsoncons::json checksumJson =
 jsoncons::jmespath::search(imageFrame,

 jmesPathExpression);
                    imageFrameIDs.mFullResolutionChecksum =
 checksumJson.as_integer<uint32_t>();

                    imageFrames.emplace_back(imageFrameIDs);
                }
            }

            result = true;
        }
        catch (const std::exception &e) {
            std::cerr << "getImageFramesForImageSet failed because " << e.what()
                      << std::endl;
        }
    }

    return result;
}

//! Routine which gets a HealthImaging image set's metadata.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The HealthImaging image set ID.
  \param versionID: The HealthImaging image set version ID, ignored if empty.
```

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

    return outcome.IsSuccess();
}
```

下载、解码和验证图像帧。

```
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;
}
```

清理资源。

```
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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的以下主题。
  - DeleteImageSet
  - getDiCom ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Startdicom ImportJob

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

index.js-编排步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
```

```
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
```

```
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js-部署资源。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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-复制 DICOM 文件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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-开始导入数据存储。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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",
  },
);
```

```
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-获取图像集 ID。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;
```

```
    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js-获取图像框 ID。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
```

```
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */
```

```
/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }
```

```
    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
  { slow: false },
);
```

verify-steps.js-验证图像框。使用AWS HealthImaging 像素数据验证库进行验证。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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",
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (/** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid}
 and sop ${sopInstanceUid}`,
          );
```

```
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(
                    `Verification tool exited with code ${code} for image set
 ${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

clean-up-steps.js-摧毁资源。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
```

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

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
```

```
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
```

```
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- 有关 API 详细信息，请参阅 AWS SDK for JavaScript API 参考中的以下主题。
  - DeleteImageSet
  - getDiCom ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Startdicom ImportJob

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

使用必要的资源创建 AWS CloudFormation 堆栈。

```
    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
```

将 DICOM 文件复制到 Amazon S3 导入桶。

```
   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.")
```

将 DICOM 文件导入 Amazon S3 数据存储。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
```

```
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def start_dicom_import_job(
        self,
        data_store_id,
        input_bucket_name,
        input_directory,
        output_bucket_name,
        output_directory,
        role_arn,
    ):
        """
        Routine which starts a HealthImaging import job.

        :param data_store_id: The HealthImaging data store ID.
        :param input_bucket_name: The name of the Amazon S3 bucket containing the
DICOM files.
        :param input_directory: The directory in the S3 bucket containing the
DICOM files.
        :param output_bucket_name: The name of the S3 bucket for the output.
        :param output_directory: The directory in the S3 bucket to store the
output.
        :param role_arn: The ARN of the IAM role with permissions for the import.
        :return: The job ID of the import.
        """

        input_uri = f"s3://{input_bucket_name}/{input_directory}/"
        output_uri = f"s3://{output_bucket_name}/{output_directory}/"
        try:
            job = self.medical_imaging_client.start_dicom_import_job(
                jobName="examplejob",
                datastoreId=data_store_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_uri,
                outputS3Uri=output_uri,
            )
```

```
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

获取由 DICOM 导入任务创建的影像集。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
        """
        Retrieves the image sets created for an import job.

        :param datastore_id: The HealthImaging data store ID
        :param import_job_id: The import job ID
        :return: List of image set IDs
        """
```

```
        import_job = self.medical_imaging_client.get_dicom_import_job(
            datastoreId=datastore_id, jobId=import_job_id
        )

        output_uri = import_job["jobProperties"]["outputS3Uri"]

        bucket = output_uri.split("/")[2]
        key = "/".join(output_uri.split("/")[3:])

        # Try to get the manifest.
        retries = 3
        while retries > 0:
            try:
                obj = self.s3_client.get_object(
                    Bucket=bucket, Key=key + "job-output-manifest.json"
                )
                body = obj["Body"]
                break
            except ClientError as error:
                retries = retries - 1
                time.sleep(3)
        try:
            data = json.load(body)
            expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
            image_sets = expression.search(data)
        except json.decoder.JSONDecodeError as error:
            image_sets = import_job["jobProperties"]

        return image_sets


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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.medical_imaging_client.get_image_set(
```

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

获取影像集的影像帧信息。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)
```

```
   def get_image_frames_for_image_set(self, datastore_id, image_set_id,
out_directory):
        """
        Get the image frames for an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param out_directory: The directory to save the file.
        :return: The image frames.
        """
        image_frames = []
        file_name = os.path.join(out_directory,
f"{image_set_id}_metadata.json.gzip")
        file_name = file_name.replace("/", "\\\\")
        self.get_image_set_metadata(file_name, datastore_id, image_set_id)
        try:
            with gzip.open(file_name, "rb") as f_in:
                doc = json.load(f_in)
            instances = jmespath.search("Study.Series.*.Instances[].*[]", doc)
            for instance in instances:
                rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
                rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)
                image_frames_json = jmespath.search("ImageFrames[][]", instance)
                for image_frame in image_frames_json:
                    checksum_json = jmespath.search(
                        "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",
                        image_frame,
                    )
                    image_frame_info = {
                        "imageSetId": image_set_id,
                        "imageFrameId": image_frame["ID"],
                        "rescaleIntercept": rescale_intercept,
                        "rescaleSlope": rescale_slope,
                        "minPixelValue": image_frame["MinPixelValue"],
                        "maxPixelValue": image_frame["MaxPixelValue"],
                        "fullResolutionChecksum": checksum_json["Checksum"],
                    }
                    image_frames.append(image_frame_info)
            return image_frames
        except TypeError:
            return {}
        except ClientError as err:
```

```
            logger.error(
                "Couldn't get image frames for image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        return image_frames


    def get_image_set_metadata(
        self, metadata_file, datastore_id, image_set_id, version_id=None
    ):
        """
        Get the metadata of an image set.

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """

        try:
            if version_id:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
            with open(metadata_file, "wb") as f:
                for chunk in
image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)

        except ClientError as err:
            logger.error(
                "Couldn't get image metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
```

下载、解码和验证图像帧。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


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

        :param file_path_to_write: The path to write the image frame's HTJ2K
 encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.medical_imaging_client.get_image_frame(
                datastoreId=datastore_id,
```

```
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    f.write(chunk)
        except ClientError as err:
            logger.error(
                "Couldn't get image frame. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def download_decode_and_check_image_frames(
        self, data_store_id, image_frames, out_directory
    ):
        """
        Downloads image frames, decodes them, and uses the checksum to validate
        the decoded images.

        :param data_store_id: The HealthImaging data store ID.
        :param image_frames: A list of dicts containing image frame information.
        :param out_directory: A directory for the downloaded images.
        :return: True if the function succeeded; otherwise, False.
        """
        total_result = True
        for image_frame in image_frames:
            image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
            self.get_pixel_data(
                image_file_path,
                data_store_id,
                image_frame["imageSetId"],
                image_frame["imageFrameId"],
            )

            image_array = self.jph_image_to_opj_bitmap(image_file_path)
            crc32_checksum = image_frame["fullResolutionChecksum"]
            # Verify checksum.
            crc32_calculated = zlib.crc32(image_array)
            image_result = crc32_checksum == crc32_calculated
            print(
```

```
                f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
{image_result }"
            )
            total_result = total_result and image_result
        return total_result

    @staticmethod
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array
```

清理资源。

```
    def destroy(self, stack):
        """
        Destroys the resources managed by the CloudFormation stack, and the
CloudFormation
        stack itself.

        :param stack: The CloudFormation stack that manages the example
resources.
        """

        print(f"\t\tCleaning up resources and {stack.name}.")
        data_store_id = None
        for oput in stack.outputs:
            if oput["OutputKey"] == "DatastoreID":
                data_store_id = oput["OutputValue"]
        if data_store_id is not None:
            print(f"\t\tDeleting image sets in data store {data_store_id}.")
            image_sets = self.medical_imaging_wrapper.search_image_sets(
```

```
                data_store_id, {}
            )
            image_set_ids = [image_set["imageSetId"] for image_set in image_sets]

            for image_set_id in image_set_ids:
                self.medical_imaging_wrapper.delete_image_set(
                    data_store_id, image_set_id
                )
                print(f"\t\tDeleted image set with id : {image_set_id}")

        print(f"\t\tDeleting {stack.name}.")
        stack.delete()
        print("\t\tWaiting for stack removal. This may take a few minutes.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_delete_complete")
        waiter.wait(StackName=stack.name)
        print("\t\tStack delete complete.")




class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
```

```
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.medical_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries


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

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

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - DeleteImageSet
  - getDiCom ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Startdicom ImportJob

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 S HealthImaging DK 为数据存储添加标签 AWS

以下代码示例显示了如何为 HealthImaging 数据存储添加标签。

Java

适用于 Java 2.x 的 SDK

标记数据存储。

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

用于标记资源的实用程序函数。

```
   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);
       }
   }
```

列出数据存储的标签。

```
               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());
               }
```

用于列出资源标签的实用程序函数。

```
   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;
    }
```

取消标记数据存储。

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

用于取消标记资源的实用程序函数。

```
    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);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的以下主题。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

标记数据存储。

```
  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);
  }
```

用于标记资源的实用程序函数。

```
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;
};
```

列出数据存储的标签。

```
  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);
  }
```

用于列出资源标签的实用程序函数。

```
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;
};
```

取消标记数据存储。

```
  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);
```

```
  }
```

用于取消标记资源的实用程序函数。

```
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;
};
```

- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的以下主题。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

标记数据存储。

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

用于标记资源的实用程序函数。

```
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
```

列出数据存储的标签。

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

用于列出资源标签的实用程序函数。

```
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"]
```

取消标记数据存储。

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

用于取消标记资源的实用程序函数。

```
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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。

- ListTagsForResource
- TagResource
- UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 SDK 为 HealthImaging 图像集添加标签 AWS

以下代码示例显示了如何为 HealthImaging 图像集添加标签。

Java

适用于 Java 2.x 的 SDK

标记映像集。

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

用于标记资源的实用程序函数。

```
   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);
        }
    }
```

列出映像集的标签。

```
                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());
                }
```

用于列出资源标签的实用程序函数。

```
    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;
    }
```

取消标记映像集。

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                Collections.singletonList("Deployment"));
```

用于取消标记资源的实用程序函数。

```
   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);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的以下主题。

- ListTagsForResource
- TagResource
- UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

标记映像集。

```
  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);
  }
```

用于标记资源的实用程序函数。

```
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;
};
```

列出映像集的标签。

```
  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);
  }
```

用于列出资源标签的实用程序函数。

```
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;
};
```

取消标记映像集。

```
  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);
  }
```

用于取消标记资源的实用程序函数。

```
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;
};
```

- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的以下主题。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Python

SDK for Python (Boto3)

标记映像集。

```
    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"})
```

用于标记资源的实用程序函数。

```
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
```

列出映像集的标签。

```
    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)
```

用于列出资源标签的实用程序函数。

```
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"]
```

取消标记映像集。

```
    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"])
```

用于取消标记资源的实用程序函数。

```
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
```

以下代码实例化对象。 MedicalImagingWrapper

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

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 HealthImaging 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

安全性 AWS HealthImaging

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在云中运行 AWS 服务的基础架构 AWS Cloud。 AWS 还为您提供可以安全使用的服务。作为AWS 合规计划合规计划合规计划合的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用的合规计划 AWS HealthImaging，请参阅按合规计划划分的范围内的AWSAWS 服务按合规计划。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用时如何应用分担责任模型 HealthImaging。以下主题向您介绍如何进行配置 HealthImaging 以满足您的安全和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 HealthImaging 资源。

主题

AWS 中的数据保护 HealthImaging

责任 AWS 共担模式适用于 AWS 中的数据保护 HealthImaging。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问

题。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的 AWS 责任共担模式和 GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API HealthImaging 或 SDK 或以其他 AWS 服务 方式使用控制台 AWS CLI、API 或 AWS SDK 的情况。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

主题

数据加密

借助 AWS HealthImaging，您可以为云中的静态数据增加一层安全保护，提供可扩展且高效的加密功能。其中包括：

- 大多数 AWS 服务都提供静态数据加密功能
- 灵活的密钥管理选项 AWS Key Management Service，包括，您可以使用这些选项来选择是 AWS 管理加密密钥还是完全控制自己的密钥。

- AWS 拥有的 AWS KMS 加密密钥
- 加密消息队列，可用于使用适用于 Amazon SQS 的服务器端加密（SSE）传输敏感数据。

此外，还 AWS 提供了 API，用于将加密和数据保护与您在 AWS 环境中开发或部署的任何服务集成在一起。

静态加密

HealthImaging 默认提供加密，以使用服务拥有的 AWS KMS 密钥保护敏感的静态客户数据。

传输中加密

HealthImaging 使用 TLS 1.2 对通过公共端点和后端服务传输的数据进行加密。

密钥管理

AWS KMS 密钥（KMS 密钥）是中的主要资源 AWS Key Management Service。您还可以生成数据密钥以供外部使用 AWS KMS。

AWS 拥有的 KMS 密钥

HealthImaging 默认使用这些密钥自动加密潜在的敏感信息，例如个人身份信息或静态私人健康信息 (PHI) 数据。 AWS 拥有的 KMS 密钥不会存储在您的账户中。它们是 KMS 密钥集合的一部分，这些密钥 AWS 拥有并管理，可在多个 AWS 账户中使用。 AWS 服务可以使用 AWS 自有的 KMS 密钥来保护您的数据。您无法查看、管理、使用 AWS 拥有的 KMS 密钥或审核其使用情况。但是无需执行任何工作或更改任何计划即可保护用于加密数据的密钥。

如果您使用 AWS 自有的 KMS 密钥，则无需支付月费或使用费，也不会计入您账户的 AWS KMS 配额。有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的 AWS 自有密钥。

客户托管式（KMS 密钥）

HealthImaging 支持使用由您创建、拥有和管理的对称客户托管 KMS 密钥来为现有的 AWS 自有加密添加第二层加密。由于您可以完全控制这层加密，因此可以执行以下任务：

- 建立和维护密钥政策、IAM Policy 和授权
- 轮换密钥加密材料
- 启用和禁用密钥政策
- 添加标签

- 创建密钥别名
- 计划删除密钥

您还可以使用 CloudTrail 来跟踪代表您 HealthImaging 发送 AWS KMS 的请求。需 AWS KMS 支付额外费用。有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的客户托管密钥。

创建客户托管密钥

您可以使用 AWS Management Console 或 AWS KMS API 创建对称的客户托管密钥。有关更多信息，请参阅AWS Key Management Service 《开发人员指南》中的创建对称 KMS 密钥。

密钥策略控制对客户托管密钥的访问。每个客户托管密钥必须只有一个密钥策略，其中包含确定谁可以使用该密钥以及如何使用该密钥的声明。创建客户托管密钥时，可以指定密钥策略。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的管理对客户托管密钥的访问。

要将客户托管密钥用于您的 HealthImaging 资源，必须在密钥策略中允许 kms: CreateGrant 操作。这会向客户托管密钥添加授权，该密钥控制对指定 KMS 密钥的访问权限，从而允许用户访问授权操作 HealthImaging 所需的权限。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》的AWS KMS中的授权。

要将客户托管的 KMS 密钥 HealthImaging 用于您的资源，必须在密钥策略中允许以下 API 操作：

- `kms:DescribeKey` 提供验证密钥所需的客户托管式密钥详细信息。这是所有操作所必需的。
- `kms:GenerateDataKey` 为所有写入操作提供对静态加密资源的访问权限。
- `kms:Decrypt` 提供对加密资源的读取或搜索操作的访问权限。
- `kms:ReEncrypt*` 提供重新加密资源的访问权限。

以下是一个策略声明示例，允许用户创建由 HealthImaging 该密钥加密的数据存储并与之交互：

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

使用客户托管 KMS 密钥时所需的 IAM 权限

使用客户托管的 KMS 密钥创建启用 AWS KMS 加密的数据存储时，创建 HealthImaging 数据存储的用户或角色需要密钥策略和 IAM 策略的权限。

有关密钥策略的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 启用 IAM Policy。

创建存储库的 IAM 用户、IAM 角色或 AWS 账户必须拥有`kms:CreateGrant`、、`kms:GenerateDataKey`kms:RetireGrant`kms:Decrypt`、和的权限`kms:ReEncrypt*`，以及必需的 AWS 权限 HealthImaging。

如何在中 HealthImaging 使用补助 AWS KMS

HealthImaging 需要获得授权才能使用您的客户托管的 KMS 密钥。当您创建使用客户托管的 KMS 密钥加密的数据存储时， HealthImaging 会通过向发送CreateGrant请求来代表您创建授权 AWS KMS。中的授权 AWS KMS 用于授予对客户账户中的 KMS 密钥的 HealthImaging 访问权限。

代表您 HealthImaging 创建的赠款不应被撤销或撤销。如果您撤销或取消授予您账户中 AWS KMS 密钥使用 HealthImaging 权限的授权，则 HealthImaging 无法访问这些数据、加密推送到数据存储的新图像资源，也无法在提取时对其进行解密。当您撤销或撤销的授予时 HealthImaging，更改会立即生效。要撤销访问权限，则应删除数据存储，而不是撤销该授权。删除数据存储后， HealthImaging 将代表您停用授权。

监控 HealthImaging 的加密密钥

使用 CloudTrail 客户托管的 KMS 密钥时，您可以使用来跟踪代表您 HealthImaging 发送的请求。AWS KMS 日志中的日志条目显示`medical-imaging.amazonaws.com`在`userAgent`字段中，以明确区分由发出的请求 HealthImaging。 CloudTrail

以下示例是CreateGrant、、和 CloudTrail 的事件 GenerateDataKeyDecrypt，用于监控DescribeKey为访问由 HealthImaging 您的客户托管密钥加密的数据而调用的 AWS KMS 操作。

以下内容显示了CreateGrant如何使用允许 HealthImaging 访问客户提供的 KMS 密钥，从而 HealthImaging 允许使用该 KMS 密钥加密所有静态客户数据。

用户无需创建自己的授权。 HealthImaging 通过向发送CreateGrant请求来代表您创建授权 AWS KMS。中的授权 AWS KMS 用于授予对客户账户中 AWS KMS 密钥的 HealthImaging 访问权限。

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

以下示例说明如何使用 GenerateDataKey 来确保用户在存储数据之前拥有加密数据的必要权限。

```
{
    "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"
}
```


以下示例显示如何 HealthImaging 调用Decrypt操作以使用存储的加密数据密钥访问加密数据。


```
{
    "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"
}
```

以下示例显示了如何 HealthImaging 使用该DescribeKey操作来验证 AWS KMS 客户拥有的 AWS KMS 密钥是否处于可用状态，以及如何帮助用户对其无法运行进行故障排除。

```
{
    "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"
}
```

了解更多信息

以下资源提供了有关静态数据加密的更多信息，其位于《AWS Key Management Service 开发人员指南》中。

- AWS KMS 概念
- 以下方面的安全最佳实践 AWS KMS

网络流量隐私

本地应用程序之间 HealthImaging 以及与 Amazon S3 HealthImaging 之间的流量都受到保护。默认情况下 HealthImaging ，和之间的流量 AWS Key Management Service 使用 HTTPS。

- AWS HealthImaging 是一项区域性服务，在美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、欧洲（爱尔兰）和亚太地区（悉尼）地区提供。
- 对于与 Amazon S3 存储桶 HealthImaging 之间的流量，传输层安全 (TLS) 会对与 A HealthImaging mazon S3 之间以及与客户应用程序之间 HealthImaging 传输的对象进行加密，您应该使用 A `aws:SecureTransport condition`mazon S3 存储桶 IAM 策略仅允许通过 HTTPS (TLS) 进行加密连接。尽管 HealthImaging 目前使用公共终端节点来访问 Amazon S3 存储桶中的数据，但这并不意味着数据会通过公共互联网。与 Amazon S3 HealthImaging 之间的所有流量均通过 AWS 网络路由，并使用 TLS 进行加密。

适用于 AWS 的 Identity and Access 管理 HealthImaging

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（拥有权限）使用 HealthImaging 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- AWS 如何 HealthImaging 与 IAM 配合使用
- AWS 基于身份的策略示例 HealthImaging

- 适用于 AWS HealthImaging 的 AWS 托管策略
- 对 AWS HealthImaging 身份和访问进行故障排除

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您所做的工作 HealthImaging。

服务用户-如果您使用 HealthImaging 服务完成工作，则管理员会为您提供所需的凭证和权限。当你使用更多 HealthImaging 功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问中的功能 HealthImaging，请参阅对 AWS HealthImaging 身份和访问进行故障排除。

服务管理员-如果您负责公司的 HealthImaging 资源，则可能拥有完全访问权限 HealthImaging。您的工作是确定您的服务用户应访问哪些 HealthImaging 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解您的公司如何将 IAM 与配合使用 HealthImaging，请参阅AWS 如何 HealthImaging 与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理访问权限 HealthImaging。要查看您可以在 IAM 中使用的 HealthImaging 基于身份的策略示例，请参阅。AWS 基于身份的策略示例 HealthImaging

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建帐户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的 为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种链接到的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文

件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Simple Storage Service（Amazon S3）存储桶策略。在支持基于资源的策略的服务中，服务管理员可

以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》 中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 - 权限边界是一个高级功能，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

AWS 如何 HealthImaging 与 IAM 配合使用

在使用 IAM 管理访问权限之前 HealthImaging，请先了解哪些可用的 IAM 功能 HealthImaging。

您可以在 AWS 中使用的 IAM 功能 HealthImaging

IAM 功能	HealthImaging 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	是
服务角色	是
服务相关角色	否

要全面了解 HealthImaging 以及其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

基于身份的策略 HealthImaging

支持基于身份的策略	是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南中的创建 IAM 策略。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

基于身份的策略示例 HealthImaging

要查看 HealthImaging 基于身份的策略的示例，请参阅。AWS 基于身份的策略示例 HealthImaging

内部基于资源的政策 HealthImaging

支持基于资源的策略	否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Simple Storage Service（Amazon S3）存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

的政策行动 HealthImaging

支持策略操作	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么 条件 下执行。

JSON 策略的 Action 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 HealthImaging 操作列表，请参阅 AWS HealthImaging 在服务授权参考中定义的操作。

正在执行的策略操作在操作前 HealthImaging 使用以下前缀：

```
AWS
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "AWS:action1",
      "AWS:action2"
         ]
```

要查看 HealthImaging 基于身份的策略的示例，请参阅。AWS 基于身份的策略示例 HealthImaging

的政策资源 HealthImaging

支持策略资源	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么条件 下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 HealthImaging 资源类型及其 ARN 的列表，请参阅《服务授权参考》 HealthImaging中的 AWS 定义的资源类型。要了解您可以使用 ARN 的操作和资源，请参阅 AWS 定义的操作。 HealthImaging

要查看 HealthImaging 基于身份的策略的示例，请参阅。AWS 基于身份的策略示例 HealthImaging

的策略条件密钥 HealthImaging

支持特定于服务的策略条件键	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看 HealthImaging 条件密钥列表，请参阅《服务授权参考》 HealthImaging中的 AWS 条件密钥。要了解您可以使用条件键的操作和资源，请参阅 AWS 定义的操作 HealthImaging。

要查看 HealthImaging 基于身份的策略的示例，请参阅。AWS 基于身份的策略示例 HealthImaging

输入的 ACL HealthImaging

支持 ACL	否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

RBAC 与 HealthImaging

支持 RBAC	是

IAM 中使用的传统授权模型称为基于角色的访问控制 (RBAC)。RBAC 根据个人的工作职能（在角色之外称为角色）来定义权限。 AWS 有关 RBAC 的更多信息，请参阅IAM 用户指南中的 ABAC 与传统 RBAC 模型的对比。

ABAC with HealthImaging

支持 ABAC（策略中的标签）	部分

> **Warning**
>
> ABAC 不是通过 `SearchImageSets` API 操作强制执行的。有权访问 `SearchImageSets` 操作的任何人都可以访问数据存储中图像集的所有元数据。

> **Note**
>
> 图像集是数据存储的子资源。要使用 ABAC，图像集必须具有与数据存储相同的标签。有关更多信息，请参阅使用 AWS 为资源添加标签 HealthImaging。

基于属性的访问控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

将临时凭证与 HealthImaging

支持临时凭证	是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的 切换到角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

的跨服务主体权限 HealthImaging

支持转发访问会话 (FAS)	是

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。策略向主体授予权限。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中触发另一个操作。在这种情况下，您必须具有执行这两个操作的权限。要查看某项操作是否需要策略中的其他依赖操作，请参阅《服务授权参考》HealthImaging中的 AWS 操作、资源和条件密钥。

HealthImaging 的服务角色

支持服务角色	是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会中断 HealthImaging 功能。只有在 HealthImaging 提供操作指导时才编辑服务角色。

的服务相关角色 HealthImaging

支持服务相关角色	否

服务相关角色是一种与服务相关联的 AWS 服务服务角色。服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

AWS 基于身份的策略示例 HealthImaging

默认情况下，用户和角色无权创建或修改 HealthImaging资源。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅 IAM 用户指南中的 创建 IAM 策略。

有关 Awesome 定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 格式，请参阅服务授权参考中的 A AWS wesome 的操作、资源和条件密钥。

主题

- 策略最佳实践
- 使用 HealthImaging控制台
- 允许用户查看他们自己的权限

策略最佳实践

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除 HealthImaging 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限 – 在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 HealthImaging控制台

要访问 AWS HealthImaging 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 HealthImaging 资源的详细信息 AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用 HealthImaging 控制台，还需要将 HealthImaging *ConsoleAccess*或*ReadOnly* AWS 托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

适用于 AWS HealthImaging 的 AWS 托管策略

AWS 托管式策略是由 AWS 创建和管理的独立策略。AWS 托管式策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，AWS 托管式策略可能不会为您的特定使用场景授予最低权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新在 AWS 托管式策略中定义的权限，则更新会影响该策略所附加到的所有主体身份（用户、组和角色）。当新的 AWS 服务启动或新的 API 操作可用于现有服务时，AWS 最有可能更新 AWS 托管式策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

主题

- AWS 托管策略：AWSHealthImagingFullAccess
- AWS 托管策略：AWSHealthImagingReadOnlyAccess
- 有关托管式策略的 AWS HealthImaging 更新

AWS 托管策略：AWSHealthImagingFullAccess

您可以将 AWSHealthImagingFullAccess 策略附加得到 IAM 身份。

此策略向所有 HealthImaging 操作授予管理权限。

```
{
    "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 托管策略：AWSHealthImagingReadOnlyAccess

您可以将 AWSHealthImagingReadOnlyAccess 策略附加得到 IAM 身份。

此策略向特定 AWS HealthImaging 操作授予只读权限。

```
{
    "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": "*"
    }]
}
```

有关托管式策略的 AWS HealthImaging 更新

查看有关 HealthImaging 的 AWS 托管策略更新的详细信息（从该服务开始跟踪这些更改开始）。要获得有关此页面更改的自动提示，请订阅释放页面上的 RSS 馈送。

更改	说明	日期
HealthImaging 开始跟踪更改	HealthImaging 为其 AWS 托管策略开启了跟踪更改。	2023 年 7 月 19 日

对 AWS HealthImaging 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 HealthImaging 和 IAM 时可能遇到的常见问题。

主题

- 我无权在以下位置执行操作 HealthImaging
- 我无权执行 iam：PassRole
- 我想允许我以外的人 AWS 账户 访问我的 HealthImaging 资源

我无权在以下位置执行操作 HealthImaging

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 AWS:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 AWS:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 AWS:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到错误消息，提示您无权执行iam:PassRole操作，则必须更新您的策略以允许您将角色传递给 HealthImaging。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为的 IAM 用户marymajor尝试使用控制台在中执行操作时，会出现以下示例错误 HealthImaging。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人 AWS 账户 访问我的 HealthImaging 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解是否 HealthImaging 支持这些功能，请参阅AWS 如何 HealthImaging 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

AWS HealthImaging 中的日志记录和监控

记录和监控是保持 AWS HealthImaging 的安全性、可靠性、可用性和性能的重要方面。AWS 提供了以下一些监控工具来记录和监控 HealthImaging、在出现错误时进行报告并适时自动采取措施。

- AWS CloudTrail 捕获由您的 AWS 账户或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Simple Storage Service（Amazon S3）存储桶。您可以标识哪些用户和账户调用了 AWS、从中发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅 AWS CloudTrail 用户指南。

- Amazon CloudWatch 实时监控您的 AWS 资源以及在 AWS 上运行的应用程序。您可以收集和跟踪指标，创建自定义的控制面板，以及设置警报以在指定的指标达到您指定的阈值时通知您或采取措施。例如，您可以具有 Amazon EC2 实例的 CloudWatch 跟踪 CPU 使用率或其他指标并且在需要时自动启动新实例。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

主题

使用记录 AWS HealthImaging API 调用 AWS CloudTrail

HealthImaging AWS 与 AWS CloudTrail一项服务集成，可记录用户、角色或 AWS 服务在中执行的操作 HealthImaging。 CloudTrail 将所有 API 调用捕获 HealthImaging 为事件。捕获的调用包括来自 HealthImaging 控制台的调用和对 HealthImaging API 操作的代码调用。如果您创建了跟踪，则可以开启向 Amazon S3 存储桶持续传输事件（包括的事件） HealthImaging。 CloudTrail如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向哪个请求发出 HealthImaging、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅《AWS CloudTrail 用户指南》。

创建跟踪

CloudTrail 在您创建账户 AWS 账户 时已为您开启。当活动发生在中时 HealthImaging，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 AWS 账户。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

> **Note**
>
> 要 HealthImaging 在中查看 AWS CloudTrail 的事件历史记录 AWS Management Console，请前往查找属性菜单，选择事件源，然后选择`medical-imaging.amazonaws.com`。

要持续记录您的 AWS 账户事件（包括的事件） HealthImaging，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存

储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

> **Note**
>
> AWS HealthImaging 支持两种类型 CloudTrail 的事件：管理事件和数据事件。管理事件是每项 AWS 服务生成的一般事件，包括 HealthImaging。默认情况下，日志记录会应用于每 HealthImaging 个 API 调用的管理事件。数据事件是可计费的，通常是为每秒交易量 (tps) 较高的 API 保留的，因此出于成本考虑，您可以选择不 CloudTrail 保存日志。
> 使用 HealthImaging，AWS API 参考中列出的所有 HealthImaging API 操作均被视为管理事件，但以下除外`GetImageFrame`。该`GetImageFrame`操作 CloudTrail 作为数据事件加载，因此必须启用。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的记录数据事件。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅CloudTrail `userIdentity`元素。

了解日志条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例显示了演示`GetDICOMImportJob`操作 HealthImaging 的 CloudTrail 日志条目。

```
{
```

```
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "XXXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
        "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/
ce6d90ba-5fba-4456-a7bc-f9bc877597c3"
        "accountId": "123456789012",
        "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "XXXXXXXXXXXXXXXXXXXXX",
                "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
                "accountId": "123456789012",
                "userName": "TestAccessRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-28T15:52:42Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-28T16:02:30Z",
    "eventSource": "medical-imaging.amazonaws.com",
    "eventName": "GetDICOMImportJob",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-
Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync
 http/Apache cfg/retry-mode/standard",
    "requestParameters": {
        "jobId": "5d08d05d6aab2a27922d6260926077d4",
        "datastoreId": "12345678901234567890123456789012"
    },
    "responseElements": null,
    "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
    "eventID": "26307f73-07f4-4276-b379-d362aa303b22",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "824333766656",
    "eventCategory": "Management"
```

```
}
```

使用 Amazon CloudWatch 监控 AWS HealthImaging

您可以使用 CloudWatch 监控 AWS HealthImaging。Amazon CloudWatch 会收集原始数据并将其处理为易读且近乎实时的指标。这些统计数据会保存 15 个月，从而使您能够使用历史信息，并能够更好地了解您的 Web 应用程序或服务的执行情况。此外，可以设置用于监测特定阈值的警报，并在达到相应阈值时发送通知或执行操作。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

> **Note**
>
> 系统会报告所有 HealthImaging API 的指标。

以下各表列出 HealthImaging 的指标和维度。每个都以用户指定数据范围的频率计数表示。

指标

指标	描述
调用计数	对 API 的调用次数。可以为账户或指定的数据存储报告此问题。 单位：计数 有效统计数据：Sum、Count 维度：操作、数据存储 ID、数据存储类型

您可以使用 AWS Management Console、AWS CLI 或 CloudWatch API 获取 HealthImaging 的指标。您可以通过某个 Amazon AWS 软件开发工具包 (SDK) 或 CloudWatch API 工具来使用 CloudWatch API。HealthImaging 控制台根据 CloudWatch API 的原始数据显示一系列图表。

您必须具有适当的 CloudWatch 权限才能使用 CloudWatch 监控 HealthImaging。有关更多信息，请参阅《CloudWatch 用户指南》中的适用于 CloudWatch 的 Identity and Access Management。

查看 HealthImaging 指标

要查看指标（CloudWatch 控制台）

1. 登录 AWS Management Console，并通过以下网址打开 CloudWatch 控制台。
2. 依次选择 指标、全部指标 和 AWS/医学成像。
3. 选择维度、指标名称，然后选择 添加到图表。
4. 选择日期范围的值。所选日期范围的指标计数将显示在该图表中。

使用 CloudWatch 创建警报

一个 CloudWatch 警报在一个指定的时间段内监控一个指标，并执行一项或多项操作：向 Amazon Simple Notification Service (Amazon SNS) 主题或自动扩缩策略发送通知。具体执行什么操作取决于在您指定的一系列时间段内指标相对于给定阈值的值。CloudWatch 警报也可以在警报状态发生变化时向您发送 Amazon SNS 消息。

CloudWatch 警报仅当状态发生变化并且已持续了您指定的时间段时才会触发操作。有关更多信息，请参阅使用 CloudWatch 警报。

AWS HealthImaging 的合规性验证

作为多个AWS合规性计划的一部分，第三方审计员将评估 AWS HealthImaging 的安全性和合规性。对于 HealthImaging，这包括 HIPAA。

有关特定合规性计划范围内的 AWS 服务列表，请参阅合规性计划范围内的 AWS 服务。有关常规信息，请参阅AWS合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参见下载 AWS Artifact 中的报告。

您使用 AWS HealthImaging 的合规性责任取决于您数据的敏感度、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- AWS合作伙伴解决方案 – 针对安全性和合归性的此自动参考部署指南讨论了架构注意事项，并提供了在 AWS 上部署基于安全性和合规性的基准环境的步骤。
- 设计符合 HIPAA 安全性和合规性要求的架构白皮书 – 此白皮书介绍公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。
- 在 GxP 系统中使用 AWS – 本白皮书提供了有关 AWS 如何处理 GxP 相关合规性和安全性的信息，并提供了在 GxP 背景下使用 AWS 服务的指导。

- AWS 合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。
- 规则评估资源 – AWS Config 评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub – 此AWS服务提供了AWS中安全状态的全面视图，可帮助您检查是否符合安全行业标准和最佳实践。

AWS HealthImaging 的故障恢复能力

AWS 全球基础设施围绕 AWS 区域 和可用区构建。AWS 区域 提供多个在物理上独立且隔离的可用区，这些可用区与延迟率低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域 和可用区的更多信息，请参阅 AWS 全球基础设施。

除了AWS 全球基础设施之外，AWS HealthImaging 还提供了多种功能，以帮助支持您的数据弹性和备份需求。

AWS HealthImaging 中的基础设施安全性

作为一项托管式服务，AWS HealthImaging 由Amazon Web Services：安全流程概览白皮书中所述的 AWS 全球网络安全程序提供保护。

您可以使用 AWS 发布的 API 调用通过网络访问 HealthImaging 。客户端必须支持传输层安全性 (TLS) 1.3 或更高版本。客户端还必须支持具有完全向前保密 (PFS) 的密码套件，例如 Ephemeral Diffie-Hellman (DHE) 或 Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您还可以使用 AWS Security Token Service (AWS STS) 生成临时安全凭证来对请求进行签名。

使用 AWS CloudFormation 创建 AWS HealthImaging 资源

AWS HealthImaging 与 AWS CloudFormation集成，后者是一项服务，可帮助您对 AWS资源进行建模和设置，这样您只需花较少的时间来创建和管理资源与基础设施。您可以创建一个描述所需的全部 AWS 资源的模板，AWS CloudFormation 将为您预置和配置这些资源。

在您使用 AWS CloudFormation 时，可重复使用您的模板来不断地重复设置您的 HealthImaging 资源。描述您的资源一次，然后在多个 AWS 账户 和区域中反复预置相同的资源。

HealthImaging 和 AWS CloudFormation 模板

要为 HealthImaging 和相关服务设置和配置资源，您必须了解AWS CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述要在 AWS CloudFormation 堆栈中调配的资源。如果您不熟悉 JSON 或 YAML，可以在 AWS CloudFormation Designer 的帮助下开始使用 AWS CloudFormation 模板。有关更多信息，请参阅 AWS CloudFormation 用户指南中的什么是 AWS CloudFormation Designer？。

AWS HealthImaging 支持使用 AWS CloudFormation 创建数据存储。有关更多信息（包括提供 HealthImaging 数据存储的 JSON 和 YAML 模板示例），请参阅 AWS CloudFormation 用户指南中的 AWS HealthImaging 资源类型参考。

了解有关 AWS CloudFormation 的更多信息

要了解有关 AWS CloudFormation 的更多信息，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation API 参考
- AWS CloudFormation 命令行界面用户指南

AWS HealthImaging 和接口 VPC 终端节点 (AWS PrivateLink)

您可以通过创建接口 VPC 终端节点在您 AWS HealthImaging 的 VPC 和之间建立私有连接。接口终端节点由一项技术提供支持，无需互联网网关AWS PrivateLink、NAT 设备、VPN 连接或 AWS Direct Connect 连接，即可使用该技术私密访问 HealthImaging API。您的 VPC 中的实例不需要公有 IP 地址即可与 HealthImaging API 通信。您的 VPC 和 VPC 之间的流量 HealthImaging 不会离开 Amazon 网络。

每个接口端点均由子网中的一个或多个弹性网络接口表示。

有关更多信息，请参阅 A mazon VPC 用户指南中的接口 VPC 终端节点 (AWS PrivateLink)。

主题

- HealthImaging VPC 终端节点的注意事项
- 为创建接口 VPC 终端节点 HealthImaging
- 为创建 VPC 终端节点策略 HealthImaging

HealthImaging VPC 终端节点的注意事项

在为设置接口 VPC 终端节点之前 HealthImaging，请务必查看 Amazon VPC 用户指南中的接口终端节点属性和限制。

HealthImaging 支持从您的 VPC 调用所有 AWS HealthImaging 操作。

为创建接口 VPC 终端节点 HealthImaging

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 HealthImaging 服务创建 VPC 终端节点。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口端点。

HealthImaging 使用以下服务名称创建 VPC 终端节点：

- com.amazonaws.*region*.medical-imaging
- com.amazonaws。 *##*。 runtime-medical-imaging
- com.amazonaws。 *##*。 dicom-medical-imaging

> **Note**
>
> 必须启用私有 DNS 才能使用 PrivateLink。

例如，您可以使用该区域的 HealthImaging 默认 DNS 名称向发出 API 请求`medical-imaging.us-east-1.amazonaws.com`。

有关更多信息，请参阅《Amazon VPC 用户指南》中的通过接口端点访问服务。

为创建 VPC 终端节点策略 HealthImaging

您可以将终端节点策略附加到控制访问权限的 VPC 终端节点 HealthImaging。该策略指定以下信息：

- 可执行操作的主体
- 可执行的操作
- 可对其执行操作的资源

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问权限。

示例：用于 HealthImaging 操作的 VPC 终端节点策略

以下是的终端节点策略示例 HealthImaging。当连接到终端节点时，此策略授予所有委托人对所有资源 HealthImaging 执行操作的访问权限。

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\":\"*\"}]}"
```

的跨账户导入 AWS HealthImaging

通过跨账户/跨区域导入，您可以将数据从位于其他支持区域的 Amazon S3 存储桶导入 HealthImaging 数据存储。您可以跨 AWS 账户、其他AWS 组织拥有的账户以及开放数据源（例如位于开放数据注册表中的 Im aging Data Commons (IDC) ）导入数据。 AWS

HealthImaging 跨账户/跨区域导入用例包括：

- 医学成像 SaaS 产品从客户账户导入 DICOM 数据
- 大型组织从多个 Amazon S3 输入存储桶中填充一个 HealthImaging 数据存储

- 研究人员在多机构临床研究中安全地共享数据

使用跨账户导入

1. Amazon S3 输入（源）存储桶拥有者必须向 HealthImaging 数据存储所有者`s3:ListBucket`授予`s3:GetObject`权限。
2. HealthImaging 数据存储所有者必须将 Amazon S3 存储桶添加到他们的 IAM 中`ImportJobDataAccessRole`。请参阅 为导入创建 IAM 角色。
3. 开始导入任务时，HealthImaging 数据存储所有者必须`inputOwnerAccountId`为 Amazon S3 输入存储桶提供。

> **Note**
>
> 通过提供`inputOwnerAccountId`，数据存储所有者可以验证输入的 Amazon S3 存储桶属于指定账户，以保持对行业标准的合规性并降低潜在的安全风险。

以下`startDICOMImportJob`代码示例包括可选`inputOwnerAccountId`参数，该参数可应用于该启动导入任务部分中的所有 AWS CLI 和 SDK 代码示例。

Java

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

    try {
        StartDicomImportJobRequest startDicomImportJobRequest =
 StartDicomImportJobRequest.builder()
                .jobName(jobName)
                .datastoreId(datastoreId)
                .dataAccessRoleArn(dataAccessRoleArn)
                .inputS3Uri(inputS3Uri)
                .outputS3Uri(outputS3Uri)
                .inputOwnerAccountId(inputOwnerAccountId)
```

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

    return "";
}
```

AWS HealthImaging 参考资料

以下参考材料可用于 AWS HealthImaging。

> **Note**
>
> 所有 HealthImaging 操作和数据类型均位于单独的参考文献中。有关更多信息，请参阅 AWS HealthImaging API 参考。

主题

DICOM 对 AWS 的支持 HealthImaging

AWS HealthImaging 支持特定的 DICOM 元素和传输语法。熟悉支持的患者、研究和系列级别的 DICOM 数据元素，因为 HealthImaging 元数据密钥是基于这些数据元素的。在开始导入之前，请验证您的医学成像数据是否符合 HealthImaging支持的传输语法和 DICOM 元素限制。

> **Note**
>
> AWS 目前 HealthImaging 不支持二进制分割图像或图标图像序列像素数据。

主题

- DICOM 元素限制
- DICOM 元数据限制

支持的 SOP 课程

借助 AWS HealthImaging，您可以导入使用任何 SOP 类 UID 编码的 DICOM P10 服务对象对 (SOP) 实例，包括停用实例和私有实例。所有私有属性也都将被保留。

元数据标准化

当您将 DICOM P10 数据导入 AWS 时 HealthImaging，它会转换为由元数据和图像框架（像素数据）组成的图像集。在转换过程中，将基于 DICOM 标准的特定版本生成 HealthImaging 元数据密钥。HealthImaging 目前基于 DI COM PS3.6 2022b 数据字典生成并支持元数据密钥。

AWS 在患者、研究和系列级别 HealthImaging 支持以下 DICOM 数据元素。

患者级别元素

> **Note**
>
> 有关每个患者级别元素的详细描述，请参阅 DICOM 数据元素注册表。

AWS HealthImaging 支持以下患者级别元素：

```
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
```

研究级别元素

> **Note**
>
> 有关每个研究级别元素的详细描述，请参阅 DICOM 数据元素注册表。

AWS HealthImaging 支持以下学习级别元素：

```
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
```

系列级别元素

> **Note**
>
> 有关每个系列级别元素的详细描述，请参阅 DICOM 数据元素注册表。

AWS HealthImaging 支持以下系列级别的元素：

```
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
```

支持的传输语法

AWS HealthImaging 导入使用下表中传输语法编码的 DICOM P10 SOP 实例。除了存储 SOP 实例外，对于使用以下 HealthImaging 传输语法编码的 SOP 实例，还可以将图像帧（像素数据）转码为 HTJ2K：

传输语法 UID	传输语法名称
1.2.840.10008.1.2	隐式 VR Endian：DICOM 的默认传输语法
1.2.840.10008.1.2.1	显式 VR Little Endian
1.2.840.10008.1.1.1.99	压缩显示 VR Little Endian
1.2.840.10008.1.2.2	显式的 VR Big Endian
1.2.840.10008.1.2.4.50	JPEG 基准式（流程 1）：有损 JPEG 8 位图像压缩的默认传输语法
1.2.840.10008.1.2.4.51	JPEG 基准式（流程 2 和 4）：有损 JPEG 12 位图像压缩（仅限流程 4）的默认传输语法
1.2.840.10008.1.2.4.57	JPEG 无损非分层结构（流程 14）
1.2.840.10008.1.2.4.70	JPEG 无损、非分层、一阶预测（进程 14 [选择值 1]）：无损 JPEG 影像压缩的默认传输语法
1.2.840.10008.1.2.4.80	JPEG-LS 无损影像压缩
1.2.840.10008.1.2.4.81	JPEG-LS 有损（近乎无损）影像压缩
1.2.840.10008.1.2.4.90	JPEG 2000 影像压缩（仅限无损）
1.2.840.10008.1.2.4.91	JPEG 2000 影像压缩
1.2.840.10008.1.2.4.201	高吞吐量 JPEG 2000 图像压缩（仅限无损）

传输语法 UID	传输语法名称
1.2.840.10008.1.2.4.202	带有 RPCL 选项的高吞吐量 JPEG 2000 图像压缩（仅限无损）
1.2.840.10008.1.2.4.203	高吞吐量 JPEG 2000 图像压缩
1.2.840.10008.1.2.5	RLE 无损

DICOM 元素限制

当您导入数据和更新图像集元数据属性时，AWS 会 HealthImaging 应用 DICOM 元素限制。下面列出了导入和元数据更新限制。

将您的医学影像数据导入 AWS 时 HealthImaging，最大长度限制适用于以下 DICOM 元素。为确保成功完成导入任务，请确认您的医学影像数据未超过长度限制。

DICOM 导入限制

HealthImaging 关键字	DICOM 关键字	DICOM 密钥	长度限制
DICOMPatientId	PatientID	(0010,0020)	min: 0, max: 64
DICOM PatientName	PatientName	(0010,0010)	min: 0, max: 256
DICOM PatientBirthDate	病人 BirthDate	(0010,0030)	min: 0, max: 18
DICOM PatientSex	PatientSex	(0010,0040)	min: 0, max: 16
DICOM UID StudyInstance	StudyInstanceUID	(0020,000D)	min: 0, max: 64
DICOM StudyId	StudyID	(0020,0010)	min: 0, max: 16
DICOM StudyDescription	StudyDescription	(0008,1030)	min: 0, max: 64
DICOM NumberOfStudy RelatedSeries	NumberOfStudyRelatedSeries	(0020,1206)	min: 0, max: 10000

HealthImaging 关键字	DICOM 关键字	DICOM 密钥	长度限制
DICOM NumberOfStudy RelatedInstances	NumberOfStudyRelated实例	(0020,1208)	min: 0, max: 10000
DICOM Accession Number	AccessionNumber	(0008,0050)	min: 0, max: 16
DICOM StudyDate	StudyDate	(0008,0020)	min: 0, max: 18
DICOM StudyTime	StudyTime	(0008,0030)	min: 0, max: 28

DICOM 元数据限制

当您使用更新 HealthImaging 元数据属性时，将应用`UpdateImageSetMetadata`以下 DICOM 约束。

- 无法更新或移除患者/研究/系列/实例级别属性的私有属性，除非更新约束同时适用于和 `updatableAttributes` `removableAttributes`
- 无法更新以下 AWS HealthImaging 生成的属性：`SchemaVersion`、`DatastoreID`、`ImageSetID`、、`PixelData`、`Checksum`、`Width`、`Height`、、
- 无法更新以下 DICOM 属性：`Tag.PixelData`、`Tag.StudyInstanceUID`、`Tag.SeriesInstanceUID`、`Tag.SOPInstanceU`
- 无法更新 VR 类型 SQ 的属性（嵌套属性）
- 无法更新多值属性
- 无法使用与 VR 类型属性不兼容的值来更新属性
- 无法更新根据 DICOM 标准不被视为有效属性的属性
- 无法跨模块更新属性。例如，如果在客户有效载荷请求的研究级别给出了患者级别的属性，则该请求可能会失效。
- 如果相关属性模块不存在于现有的 `ImageSetMetadata` 中，则无法更新属性。例如，如果现有影像集元数据中不存在带有 `seriesInstanceUID` 的序列，则不允许更新 `seriesInstanceUID` 的属性。

AWS HealthImaging 像素数据验证

AWS HealthImaging 通过检查每张影像的无损编码和解码状态来提供内置的像素数据验证。

- 当导入任务在导入图像之前捕获图像的原始像素质量状态时，图像加载过程就开始了。使用 CRC32 算法为每张图像生成唯一的不可变图像帧分辨率校验和 (IFRC)。
- IFRC 是根据每个影像帧的分辨率级别计算的。校验和值以从基本分辨率到全分辨率排序的列表形式存在于元数据文档中。
- 导入图像后，会立即对其进行解码并计算出新的 IFRC。HealthImaging 将原始图像的全分辨率 IFRC 与导入图像的新 IFRC 进行比较，以验证准确性。
- 导入任务输出日志中会捕获相应的每张影像描述性错误情况，供您查看和验证。

验证像素数据

1. 导入医学影像数据后，查看导入任务输出日志 `job-output-manifest.json` 中捕获的每张影像集的描述性成功（或错误情况）。有关 `job-output-manifest.json` 的更多信息，请参阅 了解导入任务。
2. 使用 `GetImageSetMetadata` 操作获取影像集的相关元数据。有关更多信息，请参阅获取影像集元数据。

   影像集的元数据包含有关影像帧的信息（像素数据）。`PixelDataChecksumFromBaseToFullResolution` 包含每个分辨率级别的 IFRC（校验和）。以下是作为导入任务过程一部分生成的 IFRC 元数据输出示例。

```
"ImageFrames": [{
  "ID": "678906789067890123451234512345123",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 128,
      "Height": 128,
      "Checksum": 2928338830
    },
    {
      "Width": 256,
      "Height": 256,
      "Checksum": 1362274918
    },
    {
      "Width": 512,
```

```
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
```

3. 要验证像素数据，请访问 GitHub 上的 Pixel 数据验证程序，然后按照 README.md 文件中的说明独立验证 HealthImaging 使用的各种 HTJ2K 解码库 的无损影像处理。随着数据按分辨率级别逐步加载，您可以计算原始输入数据的 IFRC，并将其与 HealthImaging 元数据中提供的相同分辨率的 IFRC 值进行比较，以验证像素数据。

适用于 AWS 的 HTJ2K 解码库 HealthImaging

在导入过程中，AWS 以高吞吐量 JPEG 2000 (HTJ2K) 无损格式对所有图像帧（像素数据）进行 HealthImaging 编码，以提供始终如一的快速图像显示和对 HTJ2K 高级功能的通用访问。由于图像帧在导入过程中以 HTJ2K 编码，因此必须先对其进行解码，然后才能在图像查看器中查看。

> **Note**
>
> HTJ2K 在 JPEG2000 标准（ISO/IEC 15444-15:2019）的第 15 部分中定义。HTJ2K 保留了 JPEG2000 的高级功能，例如分辨率可扩展性、分区、平铺、高位深度、多通道和色彩空间支持。

主题

- 解码库
- 图像查看器

解码库

根据您的编程语言，我们建议使用以下解码库来解码图像帧。

- NVIDIA nvjpeg2000：商业版、GPU 加速
- Kakadu 软件：商业版、带有 Java 和.NET 绑定的 C++
- OpenJPH：开源、C++ 和 WASM
- OpenJPEG：开源、C/C++、Java
- openjphpy：开源、Python

- pylibjpeg-openjpeg：开源、Python

图像查看器

解码后即可查看图像框。AWS HealthImaging API 操作支持各种开源图像查看器，包括：

- 开放健康影像基金会 (OHIF, Open Health Imaging Foundation)
- Cornerstone.js

AWS HealthImaging 终端节点和配额

以下主题包含有关 AWS HealthImaging 服务终端节点和配额的信息。

主题

- 服务端点
- 服务限额

服务端点

服务端点是指定作为某一 Web 服务入口点的主机和端口的 URL。每个 Web 服务请求都包含一个端点。大多数 AWS 服务都为特定区域提供终端节点，以实现更快的连接。下表列出了 AWS 的服务终端节点 HealthImaging。

区域名称	区域	端点	协议	
美国东部（弗吉尼亚州北部）	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS	
美国西部（俄勒冈州）	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS	

区域名称	区域	端点	协议	
亚太地区（悉尼）	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS	
欧洲地区（爱尔兰）	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS	

如果您使用 HTTP 请求调用 AWS HealthImaging 操作，则必须根据所调用的操作使用不同的终端节点。以下菜单列出了 HTTP 请求的可用服务端点及其支持的操作。

HTTP 请求支持的 API 操作

data store, import, tagging

以下数据存储、导入和标记操作可通过端点访问：

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- Startdicom ImportJob
- getDiCom ImportJob
- Listdicom ImportJobs
- TagResource

- ListTagsForResource
- UntagResource

image set

以下图像集操作可通过端点访问：

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging 提供了 DicomWeb 检索 WADO-RS 服务的代表。有关更多信息，请参阅 获取 DICOM 实例。

以下 DicomWeb 服务可通过端点进行访问：

```
https://dicom-medical-imaging.region.amazonaws.com
```

- 获取dicomInstance

服务限额

服务配额定义为 AWS 账户中资源、操作和项目的最大值。

> **Note**
>
> 对于可调限额，您可以使用服务限额控制台请求增加限额。有关更多信息，请参阅服务限额用户指南 中的请求增加限额。

下表列出了 AWS 的默认配额 HealthImaging。

名称	默认值	可调整	描述
每个数据存储的最大并发 CopyImageSet 请求数	每个受支持的区域：100 个	是	当前 AWS 区域中每个数据存储的最大并发 CopyImageSet 请求数
每个数据存储的最大并发 DeleteImageSet 请求数	每个受支持的区域：100 个	是	当前 AWS 区域中每个数据存储的最大并发 DeleteImageSet 请求数
每个数据存储的最大并发 UpdateImageSetMetadata 请求数	每个受支持的区域：100 个	是	当前 AWS 区域中每个数据存储的最大并发 UpdateImageSetMetadata 请求数
每个数据存储的最大并发导入任务数	ap-southeast-2：20 个 每个其他支持的区域：100 个	是	当前 AWS 区域中每个数据存储的最大并发导入任务数

名称	默认值	可调整	描述
最大数据存储	每个受支持的区域：10 个	是	当前 AWS 区域中活动数据存储的最大数量
每个 CopyImageSet 请求 ImageFrames 允许复制的最大数量	每个受支持的区域：1,000 个	是	当前 AWS 区域内每个 CopyImageSet 请求 ImageFrames 允许复制的最大数量
一个 DICOM 导入任务中的最大文件数	每个受支持的区域：5,000 个	是	当前 AWS 区域中 DICOM 导入任务中的最大文件数
一个 DICOM 导入任务中的最大嵌套文件夹数	每个受支持的区域：1 万个	否	当前 AWS 区域中 DICOM 导入任务中嵌套文件夹的最大数量
接受的最大有效载荷大小限制（以 KB 为单位）UpdateImageSetMetadata	每个支持的区域：10 KB	是	当前 AWS 区域接受的最大有效载荷大小限制（以 KB 为单位）UpdateImageSetMetadata
DICOM 导入作业中所有文件的最大大小（以 GB 为单位）	每个受支持的区域：10 GB	否	当前 AWS 区域中 DICOM 导入任务中所有文件的最大大小（以 GB 为单位）
DICOM 导入任务中，每个 DICOM P10 文件的最大大小（以 GB 为单位）	每个受支持的区域：4 GB	否	当前区域中 DICOM 导入任务中每个 DICOM P10 文件的最大大小（以 GB 为单位）AWS
ImageSetMetadata 每次导入、复制和复制的最大大小限制（以 MB 为单位）UpdateImageSet	每个受支持的区域：50 MB	是	ImageSetMetadata 每个导入、复制和 UpdateImageSet 当前 AWS 区域的最大大小限制（以 MB 为单位）

AWS HealthImaging 限制限制

您的 AWS 账户有适用于 AWS HealthImaging API 操作的限制限制。对于所有操作，如果超过节流限制，将引发 `ThrottlingException` 错误。有关更多信息，请参阅 AWS HealthImaging API 参考。

> **Note**
>
> 所有 HealthImaging API 操作的限制均可调整。要申请调整节流限制，请联系 AWS 支持中心。

下表列出了 AWS HealthImaging API 操作的限制限制。

AWS HealthImaging 限制限制

操作	限制率	节流突增
CreateDatastore	0.085 tps	1 tps
GetDatastore	10 tps	20 tps
ListDatastores	5 tps	10 tps
DeleteDatastore	0.085 tps	1 tps
Startdicom ImportJob	0.25 tps	1 tps
getDiCom ImportJob	25 tps	50 tps
Listdicom ImportJobs	10 tps	20 tps
SearchImageSets	25 tps	50 tps
GetImageSet	25 tps	50 tps
GetImageSetMetadata	50 tps	100 tps
GetImageFrame	1000 tps	2000 tps
getdicomInstance*	50 tps	100 tps
ListImageSetVersions	25 tps	50 tps

操作	限制率	节流突增
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

*代表 DicomWeb 服务

AWS HealthImaging 示例项目

AWS HealthImaging 在 GitHub 上提供了以下示例项目。

DICOM 从本地摄取到 AWS HealthImaging

一个 AWS 无服务器项目，用于部署物联网边缘解决方案，该解决方案从 DICOM DIMSE 来源（PACS、VNA、CT 扫描仪）接收 DICOM 文件并将其存储在安全的 Amazon S3 存储桶中。该解决方案将数据库中的 DICOM 文件编入索引，并将每个 DICOM 系列编入等候队列，排队等候导入 AWS HealthImaging。它由在边缘运行且由 AWS IoT Greengrass 托管的组件和在 AWS 云中运行的 DICOM 摄取管道组成。

方块等级标记 (TLM) 代理

一个使用高吞吐量 JPEG 2000 (HTJ2K) 的功能方块等级标记 (TLM) 从 AWS HealthImaging 检索影像帧的 AWS Cloud Development Kit (AWS CDK) 项目。这样可以缩短分辨率较低图像的检索时间。潜在的工作流程包括生成缩略图和逐步加载影像。

Amazon CloudFront 配送

一个 AWS 无服务器项目，用于创建带有 HTTPS 端点的 Amazon CloudFront 分配，该端点可以缓存（使用 GET）并从边缘传输影像帧。默认情况下，端点使用 Amazon Cognito JSON Web 令牌 (JWT) 对请求进行身份验证。身份验证和请求签名都是使用 Lambda @Edge 在边缘完成的。该服务是 Amazon CloudFront 的一项功能，可让您在离应用程序用户更近的地方运行代码，从而提高性能并减少延迟。无需管理基础设施。

AWS HealthImaging 查看器 UI

一个用于部署带有后端身份验证的前端 UI 的 AWS Amplify 项目，您可以使用该 UI 使用递增式解码查看存储在 AWS HealthImaging 中的影像集元数据属性和影像帧（像素数据）。您可以选择集成上面的方块等级标记 (TLM) 代理和/或 Amazon CloudFront 交付项目，以使用替代方法加载影像帧。

要查看其他示例项目，请参阅 GitHub 上的 AWS HealthImaging 示例。

HealthImaging 与 AWS SDK 一起使用

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS CLI	AWS CLI 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例
AWS Tools for PowerShell	PowerShell 代码示例工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例

SDK 文档	代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

AWS HealthImaging 发布

下表显示了 AWS HealthImaging 服务和文档的功能和更新的发布时间。

变更	说明	日期
GetDICOMInstance 用于返回 DICOM 数据	HealthImaging 提供返回给定影像集的 DICOM 第 10 部分数据（.dcm文件）的GetDICOMInstance 服务。有关更多信息，请参阅 获取 DICOM 实例。	2024年5月15日
跨账户导入	HealthImaging 支持从位于其他支持区域的 Amazon S3 存储桶导入数据。有关更多信息，请参阅 的跨账户导入 AWS HealthImaging。	2024年5月15日
图像集的搜索增强功能	HealthImaging SearchImageSets action 支持以下搜索增强功能。有关更多信息，请参阅 搜索影像集。 • 对搜索UpdatedAt 和 的额外支持 SeriesInstanceUID • 在开始时间和结束时间之间进行搜索 • 按Ascending 或对搜索结果进行排序 Descending • DICOM 系列参数在响应中返回	2024 年 4 月 3 日
导入的最大文件大小增加了	HealthImaging 支持导入任务中每个 DICOM P10 文件的最大	2024 年 3 月 6 日

	文件大小为 4 GB。有关更多信息，请参阅 服务限额。	
JPEG Lossless 和 HTJ2K 的传输语法	HealthImaging 支持以下任务导入的传输语法。有关更多信息，请参阅 支持的传输语法。 • 1.2.840.10008.1.2.4.57 — JPEG 无损非分层（流程 14） • 1.2.840.10008.1.2.4.201 — 高吞吐量 JPEG 2000 图像压缩（仅限无损） • 1.2.840.10008.1.2.4.202 — 带有 RPCL 选项图像压缩功能的高吞吐量 JPEG 2000（仅限无损） • 1.2.840.10008.1.2.4.203 — 高吞吐量 JPEG 2000 图像压缩	2024 年 2 月 16 日
测试代码示例	HealthImaging 文档提供了 Python、、 JavaScript Java AWS CLI 和 C++ 的经过测试的代码示例和 AWS 开发工具包。有关更多信息，请参阅 HealthImaging 使用 AWS SDK 的代码示例。	2023 年 12 月 19 日
导入的最大文件数增加了	HealthImaging 单个导入任务最多支持 5,000 个文件。有关更多信息，请参阅 服务限额。	2023 年 12 月 19 日
导入的嵌套文件夹	HealthImaging 单个导入任务最多支持 10,000 个嵌套文件夹。有关更多信息，请参阅 服务限额。	2023 年 12 月 1 日

更快的导入	HealthImaging 在所有支持的区域中，导入速度提高了 20 倍。有关更多信息，请参阅 服务端点。	2023 年 12 月 1 日
CloudFormation 支持	HealthImaging 支持用于配置数据存储的基础架构即代码 (IaC)。有关更多信息，请参阅 使用 AWS CloudFormation 创建 AWS HealthImaging 资源。	2023 年 9 月 21 日
正式发布	AWS HealthImaging 适用于美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、欧洲（爱尔兰）和亚太地区（悉尼）地区的所有客户。	2023 年 7 月 26 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。