aws

开发人员指南

# AWS HealthImaging

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

# AWS HealthImaging: 开发人员指南

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

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

# Table of Contents

什么是 AWS HealthImaging？ ..... 1
重要提示 ..... 2
功能 ..... 2
相关服务 ..... 3
访问 ..... 4
HIPAA ..... 4
定价 ..... 5
开始使用 ..... 6
概念 ..... 6
数据存储 ..... 6
影像集 ..... 6
元数据 ..... 7
影像帧 ..... 7
设置 ..... 7
注册参加 AWS 账户 ..... 8
创建 S3 存储桶 ..... 8
创建数据存储 ..... 8
创建 IAM 用户 ..... 9
创建一个 IAM 角色 ..... 9
安装 AWS CLI ..... 12
教程 ..... 12
管理数据存储 ..... 14
创建数据存储 ..... 14
获取数据存储属性 ..... 22
列出数据存储 ..... 30
删除数据存储 ..... 37
使用 DICOMweb ..... 45
使用 STOW-RS 存储实例 ..... 45
使用 WADO-RS 检索数据 ..... 48
检索实例 ..... 50
检索实例元数据 ..... 51
检索系列元数据 ..... 53
检索帧 ..... 54
检索批量数据 ..... 57

使用搜索数据 QIDO-RS ..... 58
适用于 dicomWeb 的搜索 API HealthImaging ..... 59
支持的 DicomWeb 查询类型 HealthImaging ..... 60
IncludeField 在 QIDO-RS 查询中使用 ..... 64
搜索研究 ..... 75
搜索系列 ..... 76
搜索实例 ..... 77
搜索研究的实例 ..... 78
搜索所有系列 ..... 80
搜索所有实例 ..... 82
OIDC 身份验证 ..... 84
令牌验证的工作原理 ..... 84
要求和设置 ..... 88
导入影像数据 ..... 99
了解导入任务 ..... 99
启动导入任务 ..... 102
获取导入任务属性 ..... 111
列出导入作业 ..... 118
访问图像集 ..... 125
了解图像集 ..... 125
搜索影像集 ..... 131
获取影像集属性 ..... 156
获取影像集元数据 ..... 162
获取影像集像素数据 ..... 173
修改图像集 ..... 181
列出影像集版本 ..... 181
更新影像集元数据 ..... 188
更新主影像集的元数据 ..... 207
将非主映像集设为主映像 ..... 208
复制图像集 ..... 209
删除一个影像集 ..... 225
标注资源 ..... 232
标记资源 ..... 232
列出资源的标签 ..... 237
取消标记资源 ..... 243
代码示例 ..... 249

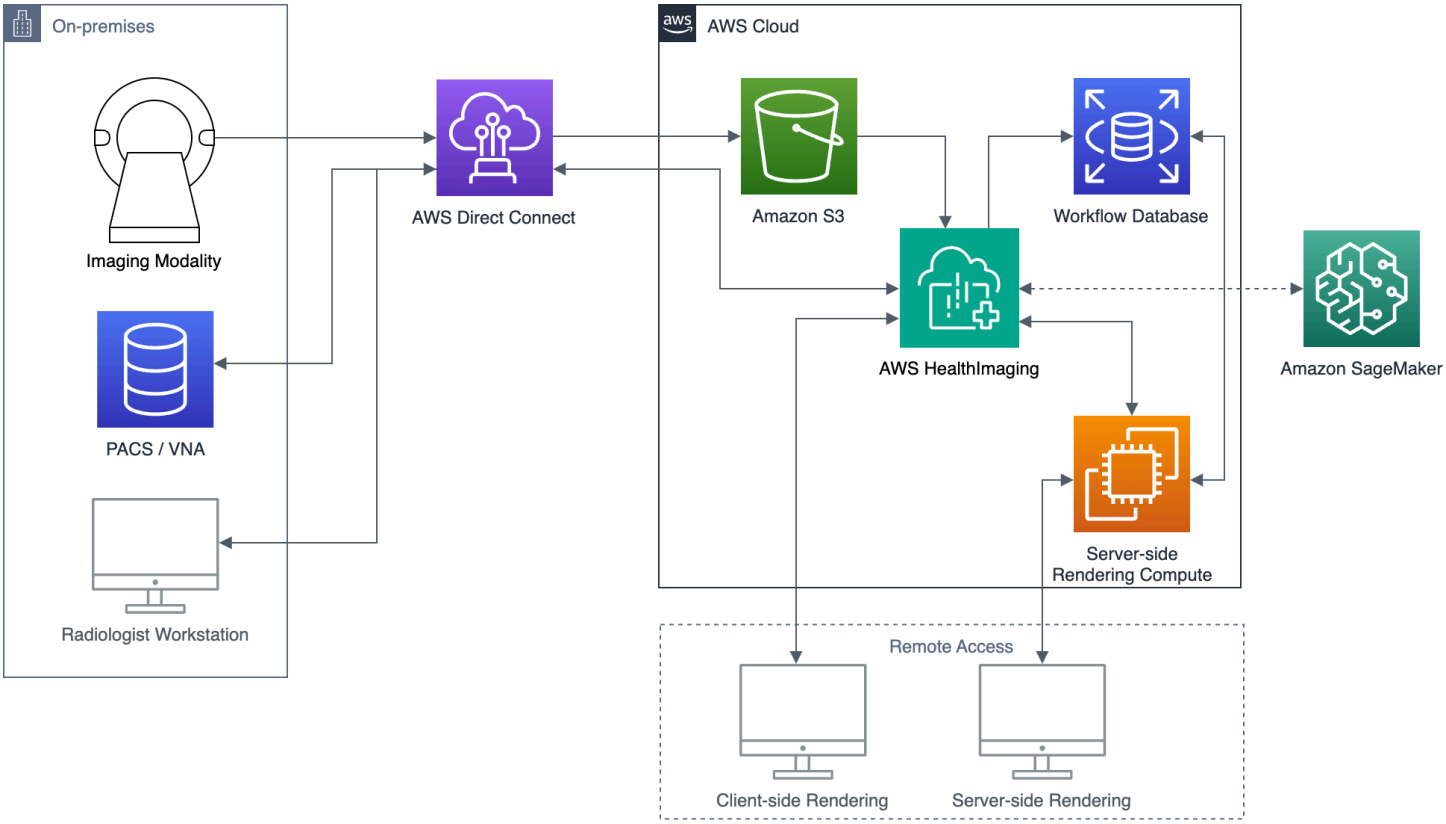
基本功能 ..... 250
你好 HealthImaging ..... 251
操作 ..... 256
场景 ..... 408
开始使用影像集和影像帧 ..... 408
标记数据存储 ..... 463
标记映像集 ..... 473
监控 ..... 484
CloudTrail （API 调用） ..... 484
创建跟踪 ..... 485
了解日志条目 ..... 486
CloudWatch （指标） ..... 487
API 指标 ..... 488
HealthImaging 指标 ..... 488
Dimensions ..... 491
访问 指标 ..... 492
设置指标 ..... 492
查看 HealthImaging 指标 ..... 492
创建警报 ..... 493
EventBridge （活动） ..... 493
HealthImaging 事件已发送至 EventBridge ..... 493
HealthImaging 事件结构和示例 ..... 495
安全性 ..... 510
数据保护 ..... 510
数据加密 ..... 511
网络流量隐私 ..... 521
身份和访问管理 ..... 521
受众 ..... 522
使用身份进行身份验证 ..... 522
使用策略管理访问 ..... 523
AWS 如何 HealthImaging 与 IAM 合作 ..... 524
基于身份的策略示例 ..... 530
AWS 托管策略 ..... 537
防止跨服务混淆代理 ..... 539
使用服务关联角色 ..... 541
问题排查 ..... 542

合规性验证 ..... 543
基础结构安全性 ..... 544
基础设施即代码 ..... 544
HealthImaging 和 CloudFormation 模板 ..... 544
了解更多关于 CloudFormation ..... 545
VPC 端点 ..... 545
VPC 端点注意事项 ..... 545
创建 VPC 端点 ..... 546
创建 VPC 端点策略 ..... 546
跨账户导入 ..... 547
恢复能力 ..... 549
参考 ..... 550
DICOM ..... 550
支持的 SOP 课程 ..... 550
元数据标准化 ..... 551
支持的传输语法 ..... 555
DICOM 元素限制 ..... 558
DICOM 元数据限制 ..... 559
HealthImaging ..... 560
端点和限额 ..... 560
节流限制 ..... 565
像素数据验证 ..... 567
警告码 ..... 569
图像帧解码库 ..... 571
示例项目 ..... 572
使用 AWS SDK ..... 573
成本优化 ..... 575
智能分层的工作原理 ..... 575
估算结构化数据存储 ..... 576
发行版 ..... 577
..... dxc

# 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。
- Long-term 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 CT X-Ray、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

- AWS 的特点 HealthImaging
- 相关 AWS 务
- 访问 AWS HealthImaging
- HIPAA 资格和数据安全
- 定价


# 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。

Developer-friendly 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。导入图像数据后，可以使用人性化的关键字而不是不熟悉的 group/element 十六进制数字来访问各个元数据属性。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-accelerated 图像解码

AWS HealthImaging 返回编码为高吞吐量 JPEG 2000 (HTJ2K) 图像的图像帧（像素数据），这是一种高级图像压缩编解码器。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，速度至少是所有其他 DICOM 传输语法的两倍。 WASM-SIMD 可以用来为零足迹的 Web 查看者带来这种极快的速度。有关更多信息，请参阅 支持的传输语法。

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。

Industry-leading 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。

服务托管 DICOM 数据层次结构

导入时，AWS HealthImaging 会自动按患者、研究和系列级 DICOM 数据元素整理 DICOM P10 数据。该服务将此 DICOM 数据组织成与 DICOM 系列相对应的影像集，从而简化了导入后的工作流程。导入新数据后，研究和系列级别的组织将保持不变。

DicomWeb API 兼容性

AWS HealthImaging 提供符合 DicomWeb 的 API，以简化集成并实现与现有应用程序的互操作性。该服务还提供云原生 API，可启用 DicomWeb 标准不支持的操作，例如元数据更新操作。

# 相关 AWS 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— 使用 IAM 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 — CloudWatch 用于观察和监控 HealthImaging 资源。
- AWS CloudTrail— CloudTrail 用于跟踪 HealthImaging 用户活动和 API 使用情况。
- AWS CloudFormation— 用于 CloudFormation 实现基础设施即代码 (IaC) 模板以在中创建资源。HealthImaging
- AWS PrivateLink— 使用 Amazon VPC 在 HealthImaging 和亚马逊虚拟私有云之间建立连接，而无需将数据暴露给互联网。
- Amazon EventBridge — 通过创建 EventBridge 将事件路由到目标的规则，用于创建可扩展、HealthImaging 事件驱动的应用程序。

# 访问 AWS HealthImaging

您可以 HealthImaging 使用 AWS 管理控制台、 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 管理控制台 以及 AWS CLI 和 AWS 软件开发工具包的代码示例。

AWS 管理控制台

AWS 管理控制台 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，请参阅构建工具 AWS。

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。有关以下内容的更多信息 AWS， U.S. 1996 年《健康保险便携性和责任法案》(HIPAA)，以及使用 AWS 服务处理、存储和传输受保护的健康信息 (PHI)，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 的合规性验证 HealthImaging。

# 定价

HealthImaging 通过智能分层帮助您自动化临床数据的生命周期管理。有关更多信息，请参阅 成本优化。

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 SDKs，必须安装和配置它们。

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
- 设置 AWS HealthImaging
- AWS HealthImaging 教程


# AWS HealthImaging 概念

以下术语和概念对您理解和使用 AWS 至关重要 HealthImaging。

概念


- 数据存储
- 影像集
- 元数据
- 影像帧


## 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成本优化。

## 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 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。 HealthImaging 尝试根据研究、系列和实例的 DICOM 层次结构组织导入的数据。成功添加

到 HealthImaging 托管层次结构的 DICOM 实例表示为主影像集。导入 DICOM P10 数据将：创建新的主影像集；如果实例已存在于主集合中，则将实例合并到现有的主影像集中；或者，如果元数据元素发生冲突，则创建一个新的非主影像集。

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 将图像集 HealthImaging 分成元数据和图像框架（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图像查看器、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。有些文件在导入过程中保留其原始传输语法编码，而另一些文件则进行转码。可以将 Amazon Web Services 数据存储配置为将无损图像帧转码为高吞吐量 JPEG 2000 (HTJ2K) 无损或 JPEG 2000 无损图像。如果图像帧采用 HTJ2 K 或 JPEG 2000 编码，则必须先对其进行解码，然后才能在图像查看器中查看。有关更多信息，请参阅 支持的传输语法、获取影像集像素数据 和 图像帧解码库。

# 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

- 注册参加 AWS 账户
- 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）

# 注册参加 AWS 账户

要开始使用 AWS，你需要一个 AWS 账户. 有关创建的信息 AWS 账户，请参阅《AWS 账户管理 参考指南》 AWS 账户中的 “入门” 。

## 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 输入存储桶存储要从该存储桶导入和 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:::amzn-s3-demo-source-bucket`
- 输出桶：`arn:aws:s3:::amzn-s3-demo-logging-bucket`

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

当您导入医学成像数据时，AWS HealthImaging 数据存储会保存您转换后的 DICOM P10 文件的结果，这些文件称为图像集。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 Well-Architected框架中的授予最低权限访问权限一致。
> 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 AWS AWS 的托管策略 HealthImaging。

## 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，该角色向 Amazon S3 存储桶授予读写权限，用于导入您的 DICOM 数据。尽管下一节的教程需要该角色，但我们建议您使用 AWS AWS 的托管策略 HealthImaging 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 能够代入该角色。

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。

   JSON

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

```
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-logging-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

3. 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。

> **Note**
>
> 此信任策略中的Condition封锁通过确保只能访问您的特定 AWS HealthImaging 数据存储，有助于防止出现混乱的代理问题。有关此安全措施的更多信息，请参阅中的Cross-service 混乱副手防范 HealthImaging。

JSON

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

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 AWS 管理控制台或 AWS SDK，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
   - 入门使用 AWS CLI

2. 在 AWS CLI `config`文件中，为管理员添加一个命名配置文件。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 的简要说明 HealthImaging和可用命令列表。

# AWS HealthImaging 教程

目标

本教程的目标是将 DICOM P10 二进制`.dcm`文件（文件）导入 AWS HealthImaging 数据存储，并将其转换为由元数据和图像框架（像素数据）组成的图像集。导入 DICOM 数据后，您可以根据自己的访问偏好使用 HealthImaging 云原生操作来访问图像集、元数据和图像框。

## 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、和使用 HealthImaging 云原生操作创建、描述、列出和删除数据存储 AWS SDKs。 AWS 管理控制台 AWS CLI

> **Note**
>
> 本章的最后一个主题是关于成本优化。将医学影像数据导入数据存储后，它会根据时间和使用情况自动在两个存储层之间移动。 HealthImaging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
- 获取数据存储属性
- 列出数据存储
- 删除数据存储


## 创建数据存储

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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。有关更多信息，请参阅 AWS HealthImaging API 参考*`CreateDatastore`*中的。创建数据存储时，您可以选择 AWS HealthImaging 用于转码和存储无损图像帧的默认传输语法。创建数据存储后，无法更改此配置。

高吞吐量 JPEG 2000 (HTJ2K)

HTJ2K（高吞吐量 JPEG 2000）是 HealthImaging 数据存储的默认存储格式。它是 JPEG 2000 标准的扩展，可显著提高编码和解码性能。在未指定 a 的情况下创建数据存储时`–lossless-storage-format`， HealthImaging 会自动使用 HTJ2 K。有关使用 HTJ2 K 创建数据存储的信息，请参阅 AWS CLI 和以下 SDKs部分。

JPEG 2000 Lossless

JPEG 2000 无损编码允许创建无需转码即可保留和检索 JPEG 2000 格式的无损图像帧的数据存储库，从而为需要 JPEG 2000 Lossless（DICOM 传输语法 UID 1.2.840.10008.1.2.4.90）的应用程序

实现更低的检索延迟，详情请参阅。支持的传输语法有关使用 JPEG 2000 无损格式创建数据存储的信息，请参阅 AWS CLI 和以下 SDKs部分。

> **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。
> - AWS 控制台支持使用默认设置创建数据存储。使用 AWS CLI 或 AWS SDK 创建–`lossless-storage-format`指定了可选内容的数据存储。

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 “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 SDKs

### 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

#### 示例 1：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。在未指定 a 的情况下创建数据存储时--lossless-storage-format，AWS HealthImaging 默认为 HTJ2K（高吞吐量 JPEG 2000）。

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

输出：

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

示例 2：采用 JPEG 2000 无损存储格式创建数据存储

使用 JPEG 2000 无损存储格式配置的数据存储将会转码并以 JPEG 2000 格式保存无损图像帧。然后，无需转码即可在 JPEG 2000 无损存储中检索图像帧。以下 create-datastore 代码示例创建配置为 JPEG 2000 无损存储格式的数据存储，其名称为 my-datastore。

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

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 的 SDK 2.x

```
    public static String createMedicalImageDatastore(MedicalImagingClient
 medicalImagingClient,
```

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

        return "";
    }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_name = 'my-datastore-name'
        oo_result = lo_mig->createdatastore( iv_datastorename =
iv_datastore_name ).
        DATA(lv_datastore_id) = oo_result->get_datastoreid( ).
        MESSAGE 'Data store created.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
```

```
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict. Data store may already exist.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migservicequotaexcdex.
        MESSAGE 'Service quota exceeded.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用CreateDatastore于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 获取数据存储属性

使用`GetDatastore`操作检索 AWS HealthImaging 数据存储属性。以下菜单提供了操作步骤 AWS 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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 SDKs

### 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

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

示例 2：获取为 JPEG2 000 配置的数据存储属性

以下 get-datastore 代码示例获取配置为 JPEG 2000 无损存储格式的数据存储的属性。

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

输出：

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

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 的 SDK 2.x

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

```
        return null;
    }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

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

- 有关 API 的详细信息，请参阅适用GetDatastore于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 列出数据存储

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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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 SDKs

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 的 SDK 2.x

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

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

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

       return null;
   }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
    TRY.
        oo_result = lo_mig->listdatastores( ).
        DATA(lt_datastores) = oo_result->get_datastoresummaries( ).
        DATA(lv_count) = lines( lt_datastores ).
        MESSAGE |Found { lv_count } data stores.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
```

```
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用ListDatastores于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

## 删除数据存储

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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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 SDKs

### 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 的 SDK 2.x

```
    public static void deleteMedicalImagingDatastore(MedicalImagingClient
medicalImagingClient,
            String datastoreID) {
        try {
```

```
            DeleteDatastoreRequest datastoreRequest =
 DeleteDatastoreRequest.builder()
                    .datastoreId(datastoreID)
                    .build();
            medicalImagingClient.deleteDatastore(datastoreRequest);
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅适用DeleteDatastore于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 在 AW DICOMweb S 中使用 HealthImaging

您可以使用的表示形式从 AWS HealthImaging 检索 DICOM 对象 DICOMweb APIs，这些对象是基于 Web 的 APIs ，符合 DICOM 医学成像标准。此功能使您能够与使用 DICOM 第 10 部分二进制文件的系统进行互操作，同时利用 HealthImaging的云原生操作。本章的重点是如何使用 HealthImaging的实现 DICOMweb APIs 来返回 DICOMweb响应。

> **重要提示**
>
> HealthImaging 将 DICOM 数据存储为影像集。使用 HealthImaging云原生操作管理和检索影像集。 HealthImaging's DICOMweb APIs 可用于返回带有 DICOMweb-conformant 响应的图像集信息。
>
> 本章中 APIs 列出的内容符合基于 Web 的医学成像DICOMweb标准。因为它们是的代表 DICOMweb APIs，所以它们不是通过 AWS CLI 和提供的 AWS SDKs。

Topic


- 使用 STOW-RS 存储实例
- 正在从中检索 DICOM 数据 HealthImaging
- 在中搜索 DICOM 数据 HealthImaging
- OIDC 身份验证适用于 DICOMweb APIs


## 使用 STOW-RS 存储实例

AWS HealthImaging 提供了DICOMweb STOW-RS APIs 用于导入数据的表示形式。使用它们 APIs 将 DICOM 数据同步存储到您的 HealthImaging 数据存储中。

下表描述了 APIs 可用于导入 HealthImaging 数据的 DICOMweb STOW-RS 表示法。

HealthImaging DICOMweb STOW-RS 的陈述 APIs

| Name | 说明 |
| --- | --- |
| `StoreDICOM` | 将一个或多个实例存储到 HealthImaging 数据存储中。 |

| Name | 说明 |
| --- | --- |
| StoreDICOMStudy | 将与指定研究实例 UID 对应的一个或多个实例存储到 HealthImaging 数据存储中。 |

使用StoreDICOM和StoreDICOMStudy操作导入的数据将组织为新的主影像集，或使用与异步导入任务相同的逻辑添加到现有的主影像集中。如果新导入的 DICOM P10 数据的元数据元素与现有的主影像集冲突，则新数据将被添加到非主影像集。

> **Note**
>
> - 这些操作支持每次请求上传最多 1GB 的 DICOM 数据。
> - API 响应将采用 JSON 格式，符合 STOW-RS 标准。 DICOMweb

发起 StorediCom 请求

1. 收集您的 AWS 区域和 DICOM P10 文件名。 HealthImaging datastoreId
2. 为表单的请求构造一个 URL：https://dicom-medical-imaging.*region*.amazonaws.com/datastore/*datastore-id*/studies
3. 例如，使用您的首选命令确定 DICOM P10 文件的内容长度。$(stat -f %z $FILENAME)
4. 准备并发送您的请求。 StoreDICOM使用带有 AWS 签名版本 4 签名协议的 HTTP POST 请求。

Example示例 1：使用操作存储 DICOM P10 文件 StoreDICOM

Shell

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

```
  --header "Content-Type: application/dicom" \
  --upload-file $FILENAME
```

Example示例 2：使用操作存储 DICOM P10 文件 StoreDICOMStudy

StorediCom 和 Store 的唯一区别DICOMStudy 是，研究实例 UID 作为参数传递给 StoreDICOMStudy，并且上传的实例必须是指定研究的成员。

Shell

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

Example示例 3：使用多部分 HTTP 有效负载存储 DICOM P10 文件

通过单个分段上传操作即可上传多个 P10 文件。以下 shell 命令演示了如何组装包含两个 P10 文件的多部分有效负载，并通过操作将其上StoreDICOM传。

Shell

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

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

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

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

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

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

# Delete the payload file
rm $FILENAME
```

# 正在从中检索 DICOM 数据 HealthImaging

AWS HealthImaging 提供了在系列和实例级别检索数据的表示形式。DICOMweb WADO-RS APIs 有了这些 APIs，就可以从数据存储中检索 DICOM 系列的所有元 HealthImaging 数据。也可以检索 DICOM 实例、DICOM 实例元数据和 DICOM 实例帧（像素数据）。 HealthImagingDICOMweb WADO-RS APIs 可以灵活地检索存储在中的数据， HealthImaging 并提供与传统应用程序的互操作性。

**重要提示**

HealthImaging 将 DICOM 数据存储为影像集。使用 HealthImaging云原生操作来管理和检索影像集。 HealthImaging's DICOMweb APIs 可用于返回带有 DICOMweb-conformant 响应的图像集信息。

本节中 APIs 列出的内容符合基于 Web 的医学成像 DICOMweb (WADO-RS) 标准。因为它们是的代表 DICOMweb APIs，所以它们不是通过 AWS CLI 和提供的 AWS SDKs。

下表描述了 APIs 可用于从中检索数据的 DICOMweb WADO-RS 的所有 HealthImaging 表示形式。HealthImaging

HealthImaging DICOMweb WADO-RS 的陈述 APIs

| Name | 说明 |
| --- | --- |
| `GetDICOMSeriesMetadata` | 通过指定与资源 UIDs 关联的研究和系列，在 HealthImaging 数据存储中检索 DICOM 系列的 DICOM 实例元数据（`.json`文件）。请参阅检索系列元数据。 |
| `GetDICOMInstance` | 通过指定与资源 UIDs 关联的系列、研究和实例，从 HealthImaging 数据存储中检索 DICOM 实例（`.dcm`文件）。请参阅检索实例。 |
| `GetDICOMInstanceMetadata` | 通过指定与资源 UIDs 关联的系列、研究和实例，从 HealthImaging 数据存储中的 DICOM 实例检索 DICOM 实例元数据（`.json`文件）。请参阅检索实例元数据。 |
| `GetDICOMInstanceFrames` | 通过指定与资源关联的系列 UID、研究 UID、实例和帧号，从 HealthImaging 数据存储中的 DICOM 实例 UIDs检索单个或批量图像帧（`multipart` 请求）。请参阅检索帧。 |

主题

- 从中获取 DICOM 实例 HealthImaging
- 从中获取 DICOM 实例元数据 HealthImaging
- 从中获取 DICOM 系列元数据 HealthImaging
- 从中获取 DICOM 实例帧 HealthImaging
- 从中获取 DICOM 批量数据 HealthImaging

# 从中获取 DICOM 实例 HealthImaging

使用GetDICOMInstance操作通过指定与资源 UIDs 关联的系列、研究和实例，从 HealthImaging 数据存储中检索 DICOM 实例（.dcm文件）。除非提供了可选的图像集参数，否则 API 将仅返回主图像集中的实例。通过将指定为查询参数，可以检索数据存储中的任何实例（来自主影像集或非主影imageSetId像集）。DICOM 数据可以通过其存储的传输语法或未压缩 (ELE) 格式进行检索。

获取 DICOM 实例 () **.dcm**

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、、、和的值构造请求studyInstanceUID的 seriesInstanceUID URL imageSetId。sopInstanceUID要查看以下示例中的整个 URL 路径，请滚动到“复制”按钮。网址的格式为：

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

4. 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`（无损图像帧的默认值）
> - 如果是`transfer-syntax=*`这样，图像帧将以存储的传输语法返回。
> - 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`-如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.202`
> - JPEG 2000 Lossless `1.2.840.10008.1.2.4.90`--如果实例存储在 HealthImaging 无损状态中。
> - JPEG 基准（流程 1）：有损 JPEG 8 位图像压缩的默认传输语法-`1.2.840.10008.1.2.4.50`-如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.50`
> - JPEG 2000 图像压缩 `1.2.840.10008.1.2.4.91`--如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.91`
> - 高吞吐量 JPEG 2000 图像压缩 `1.2.840.10008.1.2.4.203`--如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.203`
> - JPEG XL 图像压缩 `1.2.840.10008.1.2.4.112`--如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.112`
> - 存储在一个或多个 HealthImaging 以 MPEG 系列传输语法（包括 MPEG-4 AVC/H.264 and HEVC/H .265）中编码的图像帧的实例 MPEG2，可以使用相应的传输语法 UID 进行检索。例如，`1.2.840.10008.1.2.4.100`如果实例存储为 MPEG2 主配置文件主级别。

有关更多信息，请参阅支持的传输语法和AWS 的图像帧解码库 HealthImaging。

## 从中获取 DICOM 实例元数据 HealthImaging

使用`GetDICOMInstanceMetadata`操作通过指定与资源 UIDs 关联的系列、研究和实例，从 HealthImaging 数据存储中的 DICOM 实例检索元数据。除非提供了可选的图像集参数，否则 API 将仅

返回主影像集中的实例元数据。通过指定为查询参数，您可以检索数据存储中的任何实例元数据（来自主影像集或非主影imageSetId像集）。

获取 DICOM 实例元数据 () **.json**

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要查看以下示例中的整个 URL 路径，请滚动到 “复制” 按钮。网址的格式为：

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

4. 准备并发送您的请求。 GetDICOMInstanceMetadata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 实例元数据（.json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/metadata?
   imageSetId=459e50687f121185f747b67bb60d1bc8' \
     --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
     --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
     --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
     --header 'Accept: application/dicom+json'
   ```

> **Note**
>
> 元数据中指示的传输语法 UID 与中的存储传输语法 UID (StoredTransferSyntaxUID) 相匹配。 HealthImaging

## 从中获取 DICOM 系列元数据 HealthImaging

使用GetDICOMSeriesMetadata操作从数据存储中检索 DICOM 系列（.json文件）的元 HealthImaging 数据。通过指定与资源 UIDs 关联的研究和系列，您可以检索 HealthImaging 数据存储中任何主影像集的系列元数据。您可以通过提供影像集 ID 作为查询参数来检索非主影像集的系列元数据。系列元数据以DICOM JSON格式返回。

获取 DICOM 系列元数据 () **.json**

1. 收集 HealthImaging datastoreId和imageSetId参数值。
2. 使用datastoreId、studyInstanceUID、和（可选）的值为请求构造一个 URL imageSetId。seriesInstanceUID要查看以下示例中的整个 URL 路径，请滚动到 “复制” 按钮。网址的格式为：

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

3. 准备并发送您的请求。 GetDICOMSeriesMetadata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命令行工具从中获取元数据（.json文件） HealthImaging。

   Shell

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

```
  --output 'series-metadata.json'
```

使用可选imageSetId参数。

Shell

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

> **Note**
>
> - 该imageSetId参数是检索非主影像集的系列元数据所必需的。仅当指定了、、（不带imagesetID）时datastoreIdstudyInstanceUID，seriesInstanceUID该GetDICOMInstanceMetadata作才会返回主影像集的系列元数据。

## 从中获取 DICOM 实例帧 HealthImaging

使用该GetDICOMInstanceFrames操作通过指定与资源关联的系列 UID、研究 UID、实例和帧号，从 HealthImaging 数据存储中的 DICOM 实例 UIDs中检索单个或批量图像帧（multipart请求）。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帧的图像集。除非提供了可选的图像集参数，否则 API 将仅返回主图像集的实例帧。通过将指定为查询参数，可以检索数据存储中的任何实例帧（来自主影像集或非主影imageSetId像集）。

DICOM 数据可以通过其存储的传输语法或未压缩 (ELE) 格式进行检索。

获取 DICOM 实例帧 () **multipart**

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. 确定要从关联的元数据中检索的图像帧以形成frameList参数。该frameList参数是一个以逗号分隔的列表，其中包含一个或多个不重复的帧号，按任意顺序排列。例如，元数据中的第一个图像帧将是第 1 帧。

   - 单帧请求：/frames/1
   - 多帧请求：/frames/1,2,3,4

4. 使用datastoreId、、、studyInstanceUIDseriesInstanceUID、和的值构造请求sopInstanceUID的 imageSetId URL frameList。要查看以下示例中的整个 URL 路径，请滚动到 “复制” 按钮。网址的格式为：

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

5. 准备并发送您的请求。 GetDICOMInstanceFrames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命令行工具从中获取multipart响应中的图像帧 HealthImaging。

   Shell

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

> **Note**
>
> `transfer-syntax`UID 是可选的，如果不包括在内，则默认为 Explicit VR Little Endian。如果无法转码到 ELE（由于导入时有警告），则将在不进行转码的情况下返回像素。支持的传输语法包括：
>
> - 显式 VR Little Endian (ELE)-`1.2.840.10008.1.2.1`（无损图像帧的默认值）
> - 如果是`transfer-syntax=*`这样，图像帧将以存储的传输语法返回。
> - 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`-如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.202`
> - JPEG 2000 Lossless `1.2.840.10008.1.2.4.90`--如果实例存储在 HealthImaging 无损状态中。
> - JPEG 基准（流程 1）：有损 JPEG 8 位图像压缩的默认传输语法-`1.2.840.10008.1.2.4.50`-如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.50`
> - JPEG 2000 图像压缩 `1.2.840.10008.1.2.4.91`--如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.91`
> - 高吞吐量 JPEG 2000 图像压缩 `1.2.840.10008.1.2.4.203`--如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.203`
> - JPEG XL 图像压缩 `1.2.840.10008.1.2.4.112`--如果实例存储为 HealthImaging `1.2.840.10008.1.2.4.112`
> - 存储在一个或多个 HealthImaging 以 MPEG 系列传输语法（包括 MPEG-4 AVC/H.264 and HEVC/H .265）中编码的图像帧的实例 MPEG2，可以使用相应的传输语法 UID 进行检索。例如，`1.2.840.10008.1.2.4.100`如果实例存储为 MPEG2 主配置文件主级别。
> - `NotAcceptableException`如果无法根据存储的传输语法返回请求的传输语法，或者如果实例有特定的处理警告，则可能会收到 406。如果出现这种情况，请使用重试呼叫。`transfer-syntax=*`

有关更多信息，请参阅支持的传输语法和AWS 的图像帧解码库 HealthImaging。

# 从中获取 DICOM 批量数据 HealthImaging

使用GetDICOMBulkdata操作检索已在数据存储中与 DICOM 元数据分离的二进制 HealthImaging 数据。检索实例或系列元数据时，大于 1MB 的二进制属性将由BulkDataURI而不是内联值表示。您可以使用元数据响应中BulkDataURI提供的来检索 HealthImaging 数据存储中任何主影像集的二进制数据。您可以通过提供影像集 ID 作为查询参数来检索非主影像集的批量数据。

获取 DICOM 批量数据

当您从 HealthImaging DICOMweb WADO-RS 操作中检索 DICOM 元数据（例如GetDICOMInstanceMetadata或）时GetDICOMSeriesMetadata，大型二进制属性将按顺序替换为，如下所示： BulkDataURIs

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

要使用GetDICOMBulkdata操作检索 DICOM 元素，请使用以下步骤。

1. 使用以下格式的值为请求构造一个 URL：BulkDataURI

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

2. 使用 AWS 签名版本 4 签名协议以 HTTP GET 请求的形式发出您的GetDICOMBulkdata命令。以下代码示例使用curl命令行工具从主影像集中检索 DICOM 元素：

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

```
  --output 'bulkdata.bin'
```

要从非主影像集中检索 DICOM 数据元素，请提供一个ImageSetId参数：

```
curl --request GET \
  'https://dicom-medical-imaging.us-east-1.amazonaws.com/datastore/
d9a2a515ab294163a2d2f4069eed584c/
studies/1.3.6.1.4.1.5962.1.2.4.20040826285059.5457/
series/1.3.6.1.4.1.5962.1.3.4.1.20040825185059.5457/
instances/1.2.840.10008.5.1.4.1.1.7/bulkdata/b026324c6904b2a9cb4b88d6d61c81d1?
imageSetId=459e50687f121185f747b67bb60d1bc8' \
  --aws-sigv4 'aws:amz:us-east-1:medical-imaging' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --header "x-amz-security-token:$AWS_SESSION_TOKEN" \
  --header 'Accept: application/octet-stream' \
  --output 'bulkdata.bin'
```

> **Note**
>
> 该imageSetId参数是检索非主影像集的批量数据所必需的。仅当指定了datastoreId、studyInstanceUID、和（不带）时，“获取” DICOMBulkdata 操作才会返回主影像集SOPInstanceUID的批量数据。seriesInstanceUID imagesetID

# 在中搜索 DICOM 数据 HealthImaging

AWS HealthImaging 提供 DicomWeb QIDO-RS API 的表示形式，用于按患者 ID 搜索研究、系列和实例，并接收其唯一标识符以供进一步使用。 HealthImaging的 DicomWeb QIDO-RS API 可以灵活地搜索存储在中的数据， HealthImaging 并提供与传统应用程序的互操作性。

> **重要提示**
>
> HealthImaging的 dicomWeb API 可用于返回图像集信息。 QIDO-RS HealthImaging 除非另有说明，否则 DicomWeb API 仅引用主图像集。使用 HealthImaging云原生操作或 DicomWeb 操作的可选图像集参数来检索非主图像集。 HealthImaging的 dicomWeb API 可用于返回带有 DICOMweb-conformant 响应的图像集信息。
>
> 本节中列出的 API 符合基于 Web 的医学成像的 dicomWeb (QIDO-RS) 标准。它们不是通过 AWS CLI 和 AWS SDK 提供的。

# 适用于 dicomWeb 的搜索 API HealthImaging

下表描述了可用于在中搜索数据的 DicomWeb QIDO-RS API 的所有 HealthImaging 表示形式。HealthImaging

HealthImaging dicomWeb API QIDO-RS 的表示形式

| Name | 说明 |
| --- | --- |
| `SearchDICOMStudies` | 使用 GET 请求指定搜索查询元素，在中 HealthImaging 搜索 DICOM 研究。研究搜索结果以 JSON 格式返回，按上次更新排序，日期降序（从最新到最旧）。请参阅搜索研究。 |
| `SearchDICOMSeries` | 使用 GET 请求指定搜索查询元素，在中 HealthImaging 搜索 DICOM 系列。系列搜索结果以 JSON 格式返回，按`Series Number (0020, 0011)`升序排序（从最旧到最新）。请参阅搜索系列。 |
| `SearchDICOMInstances` | 使用 GET 请求指定搜索查询元素，在中 HealthImaging 搜索 DICOM 实例。实例搜索结果以 JSON 格式返回，按`Instance Number (0020, 0013)`升序排序（从最旧到最新）。请参阅搜索实例。 |
| `SearchDICOMStudyInstances` | 使用 GET 请求指定搜索查询元素，搜索研究中的 HealthImaging 所有 DICOM 实例。API 以 JSON 格式返回实例搜索结果，按上次更新日期降序（从最新到最旧）排序。请参阅搜索研究的实例。 |
| `SearchDICOMAllSeries` | HealthImaging 通过使用 GET 请求指定搜索查询元素，在数据存储中搜索所有 DICOM 系列。API 以 JSON 格式返回系列搜索结果，按`Series Number (0020,0011)` 升序排序（从最旧到最新）。请参阅搜索所有系列。 |

| Name | 说明 |
| --- | --- |
| SearchDICOMAllInstances | HealthImaging 通过使用 GET 请求指定搜索查询元素，搜索数据存储中的所有 DICOM 实例。API 以 JSON 格式返回实例搜索结果，按Instance Number (0020,0013) 升序排序（从最旧到最新）。请参阅搜索所有实例。 |

## 支持的 DicomWeb 查询类型 HealthImaging

HealthImaging 支持研究、系列和 SOP 实例级别的 QIDO-RS 分层资源查询。使用 QIDO-RS 分层搜索时 HealthImaging：

- 搜索研究会返回研究列表
- 搜索研究系列需要已知序列StudyInstanceUID并返回系列列表
- 搜索实例列表需要已知StudyInstanceUID和 SeriesInstanceUID

使用 HealthImaging，您无需在资源层次结构中指定所有父级 UID 即可运行 QIDO-RS 关系查询：

- 搜索研究的实例不需要SeriesInstanceUID.
- 在数据存储中搜索所有系列不需要StudyInstanceUID.
- 搜索数据存储中的所有实例不需要StudyInstanceUID或SeriesInstanceUID。

下表描述了支持在中搜索数据的 QIDO-RS 查询类型 HealthImaging。

HealthImaging 支持的 QIDO-RS 查询类型

| 查询类型 | 示例 |
| --- | --- |
| 属性值查询 | 在 “研究” 中搜索所有系列，其中modality=CT .<br><br>.../studies/1.3.6.1.4.1.14519.5.2.1.6279.6001.101370605276577556143013894866/series?00080060=CT |

| 查询类型 | 示例 |
| --- | --- |
| | 搜索所有分别以患者身份和研究日期为这些值的研究。<br><br>…/studies?PatientID=1123581 3&StudyDate=20130509 |
| 关键字查询 | 使用SeriesInstanceUID 关键字搜索所有系列。<br><br>.../studies/1.3.6.1.4.1.145 19.5.2.1.6279.6001.10137060 527657755614301389486 6/seri es?SeriesInstanceUID=1.3.6. 1.4.1.14519.5.2.1.6279.6001 .101370605276577556143013894868 |
| 标记查询 | 使用 group/element 表单中传递的查询参数搜索标签。<br><br>{group} {element} 比如 0020000D |
| 范围查询 | ...?Modality=CT&StudyDate=A ABBYYYY-BBCCYYYY |
| 使用和进行分页的limit结果 offset | .../studies?limit=1&offset= 0&00080020=20000101<br><br>您可以使用限制和偏移参数对搜索响应进行分页。限制的默认值为 1000，AWS HealthIma ging 终端节点和配额有关最大值，请参阅。<br><br>最大限制 = 1000，最大偏移量 = 1,000,000 |

| 查询类型 | 示例 |
| --- | --- |
| 通配符查询 | 通配符查询为使用 “*” 和 “？” 进行搜索提供了更大的灵活性。 “*” 匹配任何字符序列（包括零长度值）和 “？” 匹配任何单个字符。<br><br>在 StudyDescription 包含 “Nuclear” 的数据存储中搜索所有研究：<br><br>`.../studies?StudyDescription=*Nuclear*`<br><br>搜索所有以 “核” StudyDescription 结尾的研究：<br><br>`.../studies?StudyDescription=*Nuclear`<br><br>搜索所有以 “核” StudyDescription 开头的研究：<br><br>`.../studies?StudyDescription=Nuclear*`<br><br>搜索所有在 200965981 之后的 patienTid 完全包含任意 3 个字符的研究：<br><br>`.../studies?PatientID=200965981???` |

| 查询类型 | 示例 |
| --- | --- |
| FuzzyMatching 查询 | 通过添加模糊匹配可选查询参数，启用名称 DICOM 属性（PatientName (0010,0010)、ReferringPhysicianName (0008,0090)）的模糊匹配：<br><br>`.../studies?fuzzymatching=true&PatientName="Thomas^Albert"`<br><br>此查询对值的任何部分执行不区分大小写的前缀字匹配。 PatientName 它返回的结果 PatientName 值包括 “thomas”、“Albert”、“Thomas Albert”、“Thomas^Albert”，但不是 “hom” 或 “ber”。 |
| IncludeField 查询 | 使用`includefield` 查询参数请求默认响应集之外的其他 DICOM 属性。<br><br>按标签返回特定属性：<br><br>`.../studies?PatientID=11235813&includefield=00101081&includefield=PatientWeight`<br><br>返回所有可用属性：<br><br>`.../studies?PatientID=11235813&includefield=all`<br><br>使用虚线表示法返回序列 (SQ) 子属性：<br><br>`.../studies?PatientID=11235813&includefield=00080096.00080100`<br><br>返回私有数据元素：<br><br>`.../instances?includefield=00191001&00190010=Philips` |

| 查询类型 | 示例 |
| --- | --- |
| 关系查询 | 在模态为 CT 的数据存储中搜索所有系列：<br><br>`.../series?Modality=CT`<br><br>在研究中搜索研究日期为特定值的所有实例：<br><br>`.../studies/ StudyInstanceUID / instances?StudyDate=20240101` |

## IncludeField 在 QIDO-RS 查询中使用

`includefield`查询参数允许您请求 HealthImaging QIDO-RS 查询中设置的默认响应之外的其他 DICOM 属性。您可以在研究、系列和实例级别`includefield`上使用。

### 语法

使用以下 GET 请求格式在 QIDO-RS查询中添加其他字段：

```
GET .../studies?<query_params>&includefield=<tag_or_keyword>
GET .../studies/<StudyInstanceUID>/series?<query_params>&includefield=<tag_or_keyword>
GET .../studies/<StudyInstanceUID>/series/<SeriesInstanceUID>/instances?
<query_params>&includefield=<tag_or_keyword>
```

你可以在一个请求中指定多个`includefield`参数：

```
GET .../studies?
PatientID=11235813&includefield=00101081&includefield=00101030&includefield=00101010
```

### 包含字段支持的值

下表描述了该`includefield`参数支持的值。

支持的包含字段值

| 值类型 | 说明 | 示例 |
| --- | --- | --- |
| DICOM 标签（8 个十六进制字符） | 通过 GGGGEEEE 格式的标签请求特定的 DICOM 属性。 | `includefield=00081030` |

| 值类型 | 说明 | 示例 |
| --- | --- | --- |
| all | 请求资源级别的所有可用的 DICOM 属性。 | includefield=all |
| 虚线 SQ 路径 | <parent_tag>使用点表示法请求序列 (SQ) 属性中的特定子属性：。 <child_tag>。 | includefield=00080096.00080100 |
| 私有数据元素标签 | 请求私有标签（奇数组元素）。需要privateCreatorElement 参数。 | includefield=00191001 |
| 标准 DICOM 属性，包括批量数据 | 按标签或关键字请求特定的单个或多个属性。 | includefield=00102201 |

## 行为和规则

以下规则适用于includefield查询：

- 默认响应-如果没有includefield，则 QIDO-RS 响应仅返回标准属性集。
- includefield=all — 返回请求级别的所有可用属性。与其他allincludefield值组合使用时，all优先考虑。
- 最大标签数-一个请求最多可以包含 50 个includefield参数。
- 重复标记-重复includefield值会被删除并视为单个请求。
- 标签无效或缺失-如果请求的标签在 DICOM 数据中不存在或无效，则会在响应中默默省略该标签。其他有效includefield属性仍会返回。

## 序列 (SQ) 属性

使用点表示法请求序列 (SQ) 属性中的嵌套属性：

```
includefield=<parent_SQ_tag>.<child_tag>
```

例如，要在 (0008,0096) 范围内检索 CodeValue (0008,0100)，请执行以下 ReferringPhysicianIdentificationSequence 操作：

```
GET .../studies?PatientID=11235813&includefield=00080096.00080100
```

Multi-level 支持嵌套。例如：

```
includefield=00081115.00081199.00081150
```

## 私有标签

所有资源级别都支持私有 DICOM 数据元素（奇数组标签）。要请求私有标签，请添加`privateCreatorElement`查询参数。

使用以下语法：

```
GET .../instances?includefield=<private_tag>&<creator_tag>=<creator_name>
```

例如：

```
GET .../instances?includefield=00191001&00190010=Philips
```

以下规则适用于私有标签：

- 如果请求私有`privateCreatorElement`标签，则必须将标签和创建者姓名作为匹配参数提供。
- 如果未找到指定`privateCreatorElement`的，则会默默省略私有标签。
- 仅请求没有私有数据元素的`privateCreatorElement`标签只会返回创建者元素的名称和值。它不会返回属于该创建者区块的所有标签。

## 批量数据标签

请求的二进制值大于 1 MB 的批量数据 VR（OB、OD、OF、OL、UN、OW、OV）的 DICOM 属性将`includefield`作为原始二进制值`bulkdataURI`而不是原始二进制值返回。有关检索批量数据的更多信息，请参阅中的检索 DICOM 批量数据。 HealthImaging

## includefield=all 在每个级别返回什么？

指定后`includefield=all`，响应将包括特定资源级别的所有属性。

学习等级（包括字段=全部）

下表列出了指定时`includefield=all`在研究级别返回的所有属性。

includefield=all 的研究级别属性

| Tag | Name | VR |
|---|---|---|
| 00080005 | SpecificCharacterSet | CS |
| 00080020 | StudyDate | DA |
| 00080030 | StudyTime | TM |
| 00080050 | AccessionNumber | SH |
| 00080051 | IssuerOfAccessionNumberSequence | 平方米 |
| 00080056 | InstanceAvailability | CS |
| 00080061 | ModalitiesInStudy | CS |
| 00080062 | SOPClassesInStudy | UI |
| 00080090 | ReferringPhysicianName | PN |
| 0008009C | ConsultingPhysicianName | PN |
| 00080201 | TimezoneOffsetFromUTC | SH |
| 00081030 | StudyDescription | 哈哈 |
| 00081048 | PhysiciansOfRecord | PN |
| 00081060 | NameOfPhysiciansReadingStudy | PN |
| 00081080 | AdmittingDiagnosesDescription | 哈哈 |
| 00081190 | 检索网址 | 你的 |
| 00100010 | PatientName | PN |
| 00100020 | patienTid | 哈哈 |
| 00100021 | IssuerOfPatientID | 哈哈 |
| 00100022 | TypeOfPatientID | CS |

| Tag | Name | VR |
| --- | --- | --- |
| 00100026 | SourcePatientGroupIdentificationSequence | 平方米 |
| 00100027 | GroupOfPatientsIdentificationSequence | 平方米 |
| 00100028 | SubjectRelativePositionInImage | 美国 |
| 00100030 | PatientBirthDate | DA |
| 00100032 | PatientBirthTime | TM |
| 00100033 | PatientBirthDateInAlternativeCalendar | 哈哈 |
| 00100034 | PatientDeathDateInAlternativeCalendar | 哈哈 |
| 00100035 | PatientAlternativeCalendar | CS |
| 00100040 | PatientSex | CS |
| 00100050 | PatientInsurancePlanCodeSequence | 平方米 |
| 00100101 | PatientPrimaryLanguageCodeSequence | 平方米 |
| 00100102 | PatientPrimaryLanguageModifierCodeSequence | 平方米 |
| 00100200 | QualityControlSubject | CS |
| 00100201 | QualityControlSubjectTypeCodeSequence | 平方米 |
| 00100213 | StrainNomenclature | 哈哈 |
| 00100214 | StrainStockNumber | 哈哈 |
| 00100215 | StrainSourceRegistryCodeSequence | 平方米 |
| 00100217 | StrainSource | 哈哈 |
| 00100219 | StrainCodeSequence | 平方米 |
| 00100223 | GeneticModificationsNomenclature | 哈哈 |

| Tag | Name | VR |
| --- | --- | --- |
| 00100229 | GeneticModificationsCodeSequence | 平方米 |
| 00101001 | OtherPatientNames | PN |
| 00101005 | PatientBirthName | PN |
| 00101010 | PatientAge | AS |
| 00101020 | PatientSize | DS |
| 00101021 | PatientSizeCodeSequence | 平方米 |
| 00101022 | PatientBodyMassIndex | DS |
| 00101023 | 测量的 dap 维度 | DS |
| 00101024 | MeasuredLateralDimension | DS |
| 00101030 | PatientWeight | DS |
| 00101040 | PatientAddress | 哈哈 |
| 00101060 | PatientMotherBirthName | PN |
| 00101080 | MilitaryRank | 哈哈 |
| 00101081 | BranchOfService | 哈哈 |
| 00102000 | MedicalAlerts | 哈哈 |
| 00102110 | 过敏 | 哈哈 |
| 00102150 | CountryOfResidence | 哈哈 |
| 00102152 | RegionOfResidence | 哈哈 |
| 00102154 | PatientTelephoneNumbers | SH |
| 00102160 | EthnicGroup | SH |

| Tag | Name | VR |
| --- | --- | --- |
| 00102180 | 职业 | SH |
| 001021A0 | SmokingStatus | CS |
| 001021C0 | PregnancyStatus | 美国 |
| 001021D0 | LastMenstrualDate | DA |
| 001021F0 | PatientReligiousPreference | 哈哈 |
| 00102201 | PatientSpeciesDescription | 哈哈 |
| 00102202 | PatientSpeciesCodeSequence | 平方米 |
| 00102203 | PatientSexNeutered | CS |
| 00102210 | AnatomicalOrientationType | CS |
| 00102292 | PatientBreedDescription | 哈哈 |
| 00102293 | PatientBreedCodeSequence | 平方米 |
| 00102295 | BreedRegistrationNumber | 哈哈 |
| 00102296 | BreedRegistryCodeSequence | 平方米 |
| 00102297 | ResponsiblePerson | PN |
| 00102298 | ResponsiblePersonRole | CS |
| 001022999 | ResponsibleOrganization | 哈哈 |
| 00109431 | ExaminedBodyThickness | FL |
| 0020000D | StudyInstanceUID | UI |
| 00200010 | StudyID | SH |
| 00201206 | NumberOfStudyRelatedSeries | IS |

| Tag | Name | VR |
| --- | --- | --- |
| 00201208 | NumberOfStudyRelatedInstances | IS |
| 00321032 | RequestingPhysician | PN |
| 00321033 | RequestingService | 哈哈 |
| 00321060 | RequestedProcedureDescription | 哈哈 |
| 00321070 | RequestedContrastAgent | 哈哈 |
| 00380010 | 入学证件 | 哈哈 |
| 00380016 | RouteOfAdmissions | 哈哈 |
| 00380020 | AdmittingDate | DA |
| 00380021 | AdmittingTime | TM |
| 00380050 | SpecialNeeds | 哈哈 |
| 00380060 | ServiceEpisode身份证 | 哈哈 |
| 00380062 | ServiceEpisodeDescription | 哈哈 |
| 00380300 | CurrentPatientLocation | 哈哈 |
| 00380400 | PatientInstitutionResidence | 哈哈 |
| 00380500 | PatientState | 哈哈 |
| 00400244 | PerformedProcedureStepStartDate | DA |
| 00400245 | PerformedProcedureStepStartTime | TM |
| 00400250 | PerformedProcedureStepEndDate | DA |
| 00400251 | PerformedProcedureStepEndTime | TM |
| 00400253 | PerformedProcedureStepID | SH |

| Tag | Name | VR |
| --- | --- | --- |
| 00081032 | ProcedureCodeSequence | 平方米 |
| 00100024 | IssuerOfPatientIDQualifiersSequence | 平方米 |
| 00321034 | RequestingServiceCodeSequence | 平方米 |
| 00321064 | RequestedProcedureCodeSequence | 平方米 |
| 00401012 | ReasonForPerformedProcedureCodeSequence | 平方米 |

系列级别（包含字段=全部）

下表列出了指定时`includefield=all`返回的系列级属性。系列级别还会返回上表中列出的所有学习级别属性。

includefield=all 的系列级别属性

| Tag | Name | VR |
| --- | --- | --- |
| 00080021 | SeriesDate | DA |
| 00080031 | SeriesTime | TM |
| 00080060 | 模式 | CS |
| 00080064 | ConversionType | CS |
| 00080068 | PresentationIntentType | CS |
| 00080070 | Manufacturer | 哈哈 |
| 00080080 | InstitutionName | 哈哈 |
| 00080082 | InstitutionCodeSequence | 平方米 |
| 00081010 | StationName | SH |
| 0008103E | SeriesDescription | 哈哈 |

| Tag | Name | VR |
| --- | --- | --- |
| 0008103F | SeriesDescriptionCodeSequence | 平方米 |
| 00081040 | InstitutionalDepartmentName | 哈哈 |
| 00081041 | InstitutionalDepartmentTypeCodeSequence | 平方米 |
| 00081050 | PerformingPhysicianName | PN |
| 00081070 | OperatorsName | PN |
| 00081090 | ManufacturerModelName | 哈哈 |
| 00180010 | ContrastBolusAgent | 哈哈 |
| 00180015 | BodyPartExamined | CS |
| 00180050 | SliceThickness | DS |
| 00180088 | SpacingBetweenSlices | DS |
| 00181000 | DeviceSerialNumber | 哈哈 |
| 00181016 | SecondaryCaptureDeviceManufacturer | 哈哈 |
| 00181018 | SecondaryCaptureDeviceManufacturerModelName | 哈哈 |
| 00181019 | SecondaryCaptureDeviceSoftwareVersions | 哈哈 |
| 00181020 | SoftwareVersions | 哈哈 |
| 00181030 | ProtocolName | 哈哈 |
| 00181050 | SpatialResolution | DS |
| 00181200 | DateOfLastCalibration | DA |
| 00181201 | TimeOfLastCalibration | TM |
| 00185100 | PatientPosition | CS |

| Tag | Name | VR |
| --- | --- | --- |
| 0020000D | StudyInstanceUID | UI |
| 0020000E | SeriesInstanceUID | UI |
| 00200011 | SeriesNumber | IS |
| 00200052 | FrameOfReferenceUID | UI |
| 00200060 | 横向性 | CS |
| 00201209 | NumberOfSeriesRelatedInstances | IS |
| 00540081 | NumberOfSlices | 美国 |
| 00540101 | NumberOfTimeSlices | 美国 |
| 00541000 | SeriesType | CS |

实例级别（包含字段=全部）

在实例级别，`includefield=all`返回完整的实例级 DICOM 元数据。这包括存储在实例元数据中的 HealthImaging所有属性。返回该实例的原始 DICOM 文件中存在的每个 DICOM 标签，但像素数据属性除外。

主题

- 正在搜索 DICOM 的研究 HealthImaging
- 正在搜索 DICOM 系列 HealthImaging
- 正在中搜索 DICOM 实例 HealthImaging
- 在中搜索研究的 DICOM 实例 HealthImaging
- 正在搜索所有 DICOM 系列 HealthImaging
- 正在搜索中的所有 DICOM 实例 HealthImaging

# 正在搜索 DICOM 的研究 HealthImaging

使用 SearchDICOMStudies API 在 HealthImaging 数据存储中搜索 DICOM 研究。您可以 HealthImaging 通过构造包含支持的 DICOM 数据元素（属性）的 URL 来搜索 DICOM 研究。研究搜索结果以 JSON 格式返回，按上次更新排序，日期降序（从最新到最旧）。

搜索 DICOM 研究报告

1. 收集 HealthImaging region和datastoreId价值。有关更多信息，请参阅 获取数据存储属性。
2. 为请求构建 URL，包括所有适用的研究元素。要查看以下示例中的整个 URL 路径，请滚动到 “复制” 按钮。网址的格式为：

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

的研究要素 **SearchDICOMStudies**

| DICOM 元素标签 | DICOM 元素名称 |
|---|---|
| (0008,0020) | Study Date |
| (0008,0030) | StudyTime |
| (0008,0050) | Accession Number |
| (0008,0061) | Modalities in Study |
| (0008,0090) | Referring Physician Name |
| (0008,1030) | Study Description |
| (0010,0010) | Patient Name |
| (0010,0020) | Patient ID |
| (0010,0030) | Patient BirthDate |
| (0010,0032) | Patient BirthTime |
| (0020,000D) | Study Instance UID |

| DICOM 元素标签 | DICOM 元素名称 |
| --- | --- |
| (0020,0010) | Study ID |

3. 准备并发送您的请求。SearchDICOMStudies使用带有AWS 签名版本 4 签名协议的 HTTP GET 请求。以下示例使用curl命令行工具搜索有关 DICOM 研究的信息。

   curl

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

   研究搜索结果以 JSON 格式返回，按上次更新排序，日期降序（从最新到最旧）。

## 正在搜索 DICOM 系列 HealthImaging

使用 SearchDICOMSeries API 在 HealthImaging 数据存储中搜索 DICOM 系列。您可以 HealthImaging 通过构造包含支持的 DICOM 数据元素（属性）的 URL 来搜索 DICOM 系列。系列搜索结果以 JSON 格式返回，按升序排序（从最旧到最新）。

搜索 DICOM 系列

1. 收集 HealthImaging region和datastoreId价值。有关更多信息，请参阅 获取数据存储属性。
2. 收集StudyInstanceUID价值。有关更多信息，请参阅 获取影像集元数据。
3. 为请求构建 URL，包括所有适用的系列元素。要查看以下示例中的整个 URL 路径，请滚动到 “复制” 按钮。网址的格式为：

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

的系列元素 **SearchDICOMSeries**

| DICOM 元素标签 | DICOM 元素名称 |
| --- | --- |
| (0008,0060) | Modality |
| (0020,000E) | Series Instance UID |

4. 准备并发送您的请求。 SearchDICOMSeries使用带有AWS 签名版本 4 签名协议的 HTTP GET 请求。以下示例使用curl命令行工具搜索 DICOM 系列信息。

   curl

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

   系列搜索结果以 JSON 格式返回，按Series Number (0020,0011)升序排序（从最旧到最新）。

## 正在中搜索 DICOM 实例 HealthImaging

使用 SearchDICOMInstances API 在 HealthImaging 数据存储中搜索 DICOM 实例。您可以 HealthImaging 通过构造包含支持的 DICOM 数据元素（属性）的 URL 来在中搜索 DICOM 实例。实例结果以 JSON 格式返回，按升序排序（从最旧到最新）。

搜索 DICOM 实例

1. 收集 HealthImaging region和datastoreId价值。有关更多信息，请参阅 获取数据存储属性。
2. 收集StudyInstanceUID和的值SeriesInstanceUID。有关更多信息，请参阅 获取影像集元数据。
3. 为请求构建 URL，包括所有适用的搜索元素。要查看以下示例中的整个 URL 路径，请滚动到 “复制” 按钮。网址的格式为：

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

的实例元素 **SearchDICOMInstances**

| DICOM 元素标签 | DICOM 元素名称 |
| --- | --- |
| (0008,0016) | SOP Class UID |
| (0008,0018) | SOP Instance UID |
| (0008,1196) | WarningReason |

HealthImaging 使用 DICOM 元素 (0008,1196) 来保留导入警告代码。可以在实例级别搜索导入警告代码。可以使用通配符或特定的警告代码搜索导入警告代码。请参阅HealthImaging 警告码。

4. 准备并发送您的请求。 SearchDICOMInstances使用带有AWS 签名版本 4 签名协议的 HTTP GET 请求。以下示例使用curl命令行工具搜索有关 DICOM 实例的信息。

   curl

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

   实例搜索结果以 JSON 格式返回，按Instance Number (0020,0013)升序排序（从最旧到最新）

## 在中搜索研究的 DICOM 实例 HealthImaging

使用 SearchDICOMStudyInstances API 在 HealthImaging 数据存储中搜索研究中的所有 DICOM 实例。使用支持的 DICOM 数据元素（属性）构造请求网址以筛选结果。API 以 JSON 格式返回实例搜索结果，按上次更新日期降序（从最新到最旧）排序。

搜索研究的 DICOM 实例

1. 收集 HealthImaging `region`和`datastoreId`价值。有关更多信息，请参阅 获取数据存储属性。
2. 收集`StudyInstanceUID`价值。有关更多信息，请参阅 获取影像集元数据。
3. 为请求构建 URL，包括所有适用的搜索元素。网址的格式为：

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

下表列出了支持的 DICOM 数据元素（属性）。

的实例元素 **`SearchDICOMStudyInstances`**

| DICOM 元素标签 | DICOM 元素名称 |
|---|---|
| (0008,0020) | Study Date |
| (0008,0030) | Study Time |
| (0008,0050) | Accession Number |
| (0008,0061) | Modalities in Study |
| (0008,0060) | Modality |
| (0008,0090) | Referring Physician Name |
| (0008,1030) | Study Description |
| (0010,0010) | Patient Name |
| (0010,0020) | Patient ID |
| (0010,0030) | Patient BirthDate |
| (0010,0032) | Patient BirthTime |
| (0020,000D) | Study Instance UID |
| (0020,0010) | Study ID |

| DICOM 元素标签 | DICOM 元素名称 |
| --- | --- |
| (0020,000E) | Series Instance UID |
| (0020,0011) | Series Number |
| (0040,0244) | Performed Procedure Step Start Date |
| (0040,0245) | Performed Procedure Step Start Time |
| (0008,0016) | SOP Class UID |
| (0008,0018) | SOP Instance UID |

4. 准备并发送您的请求。 SearchDICOMStudyInstances使用使用签AWS 名版本 4 签名的 HTTP GET 请求。以下示例使用curl命令行工具。*region*替换为您所在 AWS 的地区。*datastoreId*替换为您的数据存储 ID。*StudyInstanceUID*替换为研究的 UID。

   curl

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

## 正在搜索所有 DICOM 系列 HealthImaging

使用 SearchDICOMAllSeries API 在 HealthImaging数据存储中搜索所有 DICOM 系列。使用支持的 DICOM 数据元素（属性）构造请求网址以筛选结果。API 以 JSON 格式返回系列搜索结果，按Series Number (0020,0011)升序排序（从最旧到最新）。

要搜索所有 DICOM 系列

1. 收集 HealthImaging region和datastoreId价值。有关更多信息，请参阅 获取数据存储属性。
2. 为请求构建 URL，包括所有适用的系列元素。网址的格式为：

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

下表列出了支持的 DICOM 数据元素（属性）。

的系列元素 **SearchDICOMAllSeries**

| DICOM 元素标签 | DICOM 元素名称 |
|---|---|
| (0008,0020) | Study Date |
| (0008,0030) | Study Time |
| (0008,0050) | Accession Number |
| (0008,0061) | Modalities in Study |
| (0008,0060) | Modality |
| (0008,0090) | Referring Physician Name |
| (0008,1030) | Study Description |
| (0010,0010) | Patient Name |
| (0010,0020) | Patient ID |
| (0010,0030) | Patient BirthDate |
| (0010,0032) | Patient BirthTime |
| (0020,000D) | Study Instance UID |
| (0020,0010) | Study ID |
| (0020,000E) | Series Instance UID |
| (0020,0011) | Series Number |
| (0040,0244) | Performed Procedure Step Start Date |

| DICOM 元素标签 | DICOM 元素名称 |
| --- | --- |
| (0040,0245) | Performed Procedure Step Start Time |

3. 准备并发送您的请求。 SearchDICOMAllSeries使用使用签AWS 名版本 4 签名的 HTTP GET 请求。以下示例使用curl命令行工具。*region*替换为您所在 AWS 的地区。*datastoreId*替换为您的数据存储 ID。

   curl

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

## 正在搜索中的所有 DICOM 实例 HealthImaging

使用 SearchDICOMAllInstances API 在 HealthImaging 数据存储中搜索所有 DICOM 实例。构造一个包含支持的 DICOM 数据元素（属性）的 URL 以筛选结果。API 以 JSON 格式返回实例搜索结果，按Instance Number (0020,0013)升序排序（从最旧到最新）。

搜索所有 DICOM 实例

1. 收集 HealthImaging region和datastoreId价值。有关更多信息，请参阅 获取数据存储属性。
2. 为请求构建 URL，包括所有适用的搜索元素。网址的格式为：

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

   下表列出了支持的 DICOM 数据元素（属性）。

的实例元素 **SearchDICOMAllInstances**

| DICOM 元素标签 | DICOM 元素名称 |
| --- | --- |
| (0008,0020) | Study Date |
| (0008,0030) | Study Time |
| (0008,0050) | Accession Number |
| (0008,0061) | Modalities in Study |
| (0008,0060) | Modality |
| (0008,0090) | Referring Physician Name |
| (0008,1030) | Study Description |
| (0010,0010) | Patient Name |
| (0010,0020) | Patient ID |
| (0010,0030) | Patient BirthDate |
| (0010,0032) | Patient BirthTime |
| (0020,000D) | Study Instance UID |
| (0020,0010) | Study ID |
| (0020,000E) | Series Instance UID |
| (0020,0011) | Series Number |
| (0040,0244) | Performed Procedure Step Start Date |
| (0040,0245) | Performed Procedure Step Start Time |
| (0008,0016) | SOP Class UID |
| (0008,0018) | SOP Instance UID |

3. 准备并发送您的请求。 SearchDICOMAllInstances使用使用签AWS 名版本 4 签名的 HTTP GET 请求。以下示例使用curl命令行工具。替换*region*为你所在 AWS 的地区。*datastoreId*替换为您的数据存储 ID。

   curl

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

# OIDC 身份验证适用于 DICOMweb APIs

除了现有的AWS 签名版本 4 (sigv4) 身份验证外，AWS 还 HealthImaging 支持使用 OpenID Connect (OIDC) 对 DICOMweb API 请求进行基于 OAuth 2.0 的身份验证。OIDC 使您能够 HealthImaging 直接与外部身份提供商 (IdPs) 集成，并使您能够为基于标准的应用程序提供通过 HealthImaging DICOMweb 端点访问您的医学成像数据的权限，而无需每个应用程序都具有凭证。 AWS

主题

- 使用 Lambda 授权方进行自定义令牌验证
- 为 OIDC 身份验证设置一个 AWS Lambda 授权机构

## 使用 Lambda 授权方进行自定义令牌验证

HealthImaging 通过使用 Lambda 授权方的架构实现 OIDC 支持，允许客户实现自己的令牌验证逻辑。这种设计使您可以灵活控制令牌的验证方式以及访问决策的执行方式，同时适应了兼容 OIDC 的身份提供商 (IdPs) 的多样化格局和不同的令牌验证方法。

### 身份验证流程

以下是身份验证的高级工作原理：

1. 客户端调用 DICOMweb API：您的应用程序使用您选择的 OIDC 身份提供商进行身份验证并收到签名 ID 令牌 (JWT)。对于每个 DICOMweb HTTP 请求，客户端必须在授权标头中包含 OIDC 访问令

牌（通常是持有者令牌）。在请求到达您的数据之前，从传入的请求中 HealthImaging 提取此令牌并调用您配置的 Lambda 授权方。

a. 标题通常遵循以下格式:`Authorization: Bearer <token>`.

2. 初始验证： HealthImaging 验证访问令牌声明，以便快速拒绝任何明显无效或过期的令牌，而无需不必要地调用 Lambda 函数。 HealthImaging 在调用 Lambda 授权方之前，对访问令牌中的某些标准声明进行初步验证：

   a. `iat`（发行时间）： HealthImaging 检查代币的发行时间是否在可接受的范围内。

   b. `exp`（到期时间）： HealthImaging 验证令牌是否未过期。

   c. `nbf`（Not Before Time）：如果存在，则 HealthImaging 确保令牌在其有效开始时间之前未被使用。

3. HealthImaging 调用 Lambda 授权方：如果初始声明验证通过， HealthImaging 则将进一步的令牌验证委托给客户配置的 L ambda 授权方函数。 HealthImaging 将提取的令牌和其他相关请求信息传递给 Lambda 函数。Lambda 函数验证令牌的签名和声明。

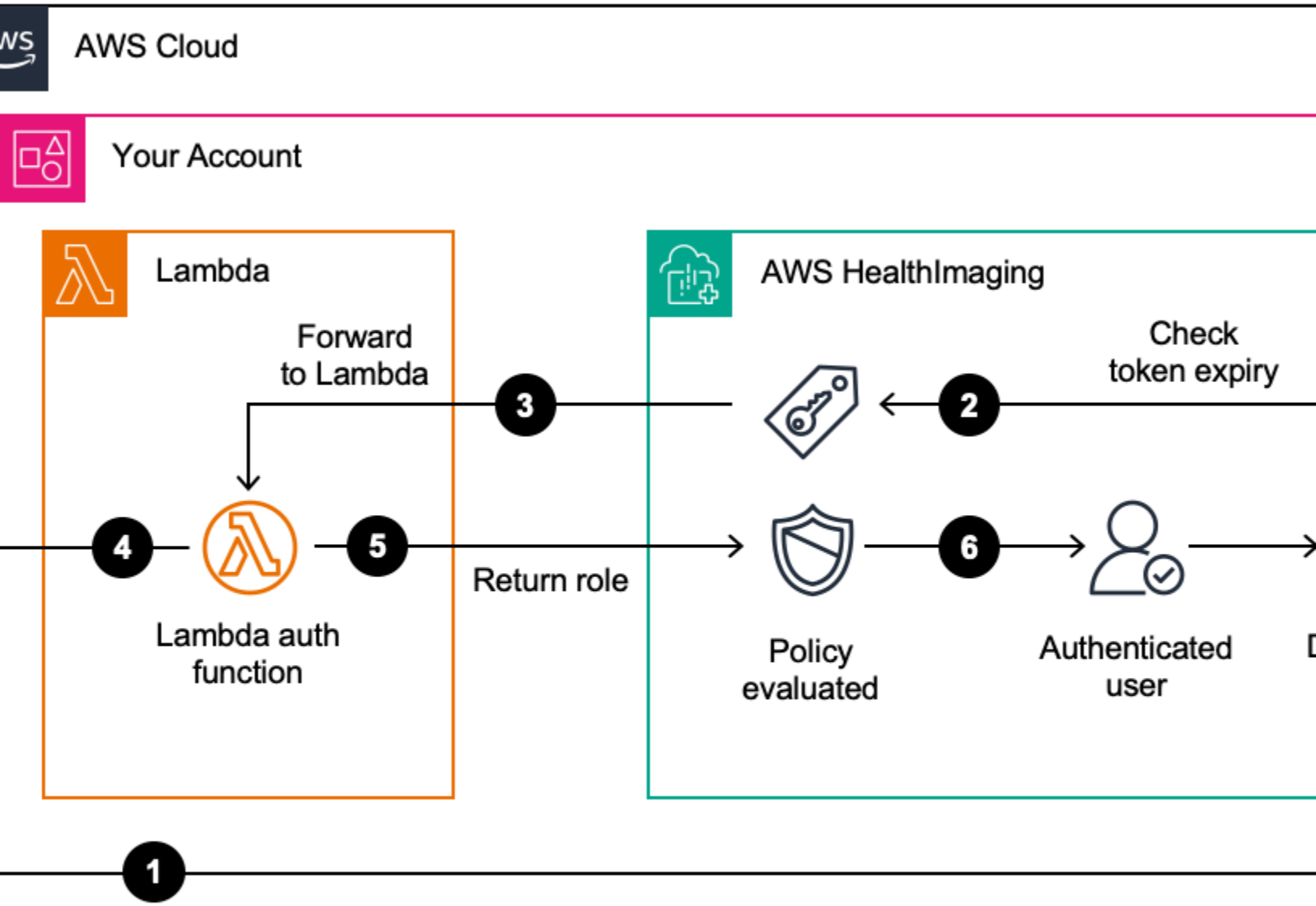
4. 向@@ 身份提供者验证：Lambda 包含自定义代码，用于检查 ID 令牌签名，执行更广泛的令牌验证（例如颁发者、受众、自定义声明），并在必要时对 IdP 验证这些声明。

5. A@@ uthorizer 返回访问策略：成功验证后，Lambda 函数将确定经过身份验证的用户的相应权限。然后，Lambda 授权方返回一个 IAM 角色的亚马逊资源名称 (ARN)，该角色代表要授予的权限集。

6. 请求执行：如果代入的 IAM 角色具有必要的权限 HealthImaging ，则继续返回请求的 DICOMWeb 资源。如果权限不足，则 HealthImaging 拒绝请求并返回相应的错误响应错误（即 403 Forbidden）。

> **Note**
>
> 授权方 lambda 函数不是由 AWS HealthImaging 服务管理的。它在你的 AWS 账户中执行。向客户收取函数调用和执行时间的费用与其 HealthImaging 费用分开收费。

## 架构概述




使用 Lambda 授权机构进行 OIDC 身份验证工作流程

## 先决条件

### 访问令牌要求

HealthImaging 要求访问令牌采用 JSON 网络令牌 (JWT) 格式。许多身份提供商 (IDPs) 本身就提供这种令牌格式，而其他身份提供者则允许您选择或配置访问令牌表单。在继续集成之前，请确保您选择的 IDP 可以发放 JWT 代币。

#### 标记格式

访问令牌必须采用 JWT（JSON 网络令牌）格式

#### 所需声明

`exp`（到期时间）

必填声明，用于指定令牌何时失效。

- 必须晚于 UTC 中的当前时间
- 表示令牌何时失效

`iat`（发布于）

必填声明，用于指定令牌的发行时间。

- 必须早于 UTC 中的当前时间
- 不得早于当前时间（UTC）前 12 小时
- 这实际上强制令牌的最大生命周期为 12 小时

`nbf`（不在时间之前）

可选声明，用于指定令牌的最早使用时间。

- 如果存在，将由以下人员进行评估 HealthImaging
- 指定在此之前不得接受令牌的时间

### Lambda 授权方响应时间要求

HealthImaging 对 Lambda 授权方响应强制执行严格的计时要求，以确保最佳 API 性能。您的 Lambda 函数必须在 1 秒钟内返回。

### 最佳实践

优化令牌验证

- 尽可能缓存 JWKS（JSON 网络密钥集）
- 尽可能缓存有效的访问令牌
- 尽量减少对身份提供商的网络呼叫
- 实现高效的代币验证逻辑

Lambda 配置

- 基于 Python 和 Node.js 的函数的初始化速度通常更快
- 减少要加载的外部库数量
- 配置适当的内存分配以确保一致的性能
- 使用 CloudWatch 指标监控执行时间

### OIDC 身份验证启用

- 只有在创建新的数据存储时才能启用 OIDC 身份验证
- API 不支持为现有数据存储启用 OIDC
- 要在现有数据存储上启用 OIDC，客户必须联系 Support AWS

## 为 OIDC 身份验证设置一个 AWS Lambda 授权机构

本指南假设您已经将所选的身份提供商 (IdP) 配置为提供与 HealthImaging OIDC 身份验证功能要求兼容的访问令牌。

### 1. 配置用于 DICOMWeb API 访问的 IAM 角色

在配置 Lambda 授权方之前，请创建 HealthImaging 要在处理 DICOMWeb API 请求时代入的 IAM 角色。授权方 Lambda 函数在成功进行令牌验证后返回其中一个角色 ARN， HealthImaging 允许使用适当的权限执行请求。

1. 创建定义所需的 DICOMWeb API 权限的 IAM 策略。有关可用权限，请参阅 HealthImaging 文档的 DICOMweb “使用” 部分。

2. 创建可执行以下操作的 IAM 角色：
   - 附上这些政策
   - 包括允许 AWS HealthImaging 服务委托人 (`medical-imaging.amazonaws.com`) 担任这些角色的信任关系。

以下是允许关联角色访问 HealthImaging DICOMWeb 只读 API 的策略示例：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "MedicalImagingDicomWebOperations",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMInstances",
                "medical-imaging:GetImageSetMetadata",
                "medical-imaging:GetDICOMSeriesMetadata",
                "medical-imaging:SearchDICOMStudies",
                "medical-imaging:GetDICOMBulkdata",
                "medical-imaging:SearchDICOMSeries",
                "medical-imaging:GetDICOMInstanceMetadata",
                "medical-imaging:GetDICOMInstance",
                "medical-imaging:GetDICOMInstanceFrames"
            ],
            "Resource": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/datastore-123"
        }
    ]
}
```

以下是应与角色关联的信任关系策略的示例：

JSON

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

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

您将在下一步中创建的 Lambda 授权者可以评估令牌声明并返回相应角色的 ARN。然后，AWS HealthImaging 将模拟此角色以相应的权限执行 DICOMWeb API 请求。

例如：

- 具有 “管理员” 声明的令牌可能会返回具有完全访问权限的角色的 ARN
- 具有 “读者” 声明的令牌可能会为具有只读访问权限的角色返回 ARN
- 具有 “department_A” 声明的令牌可能会返回特定于该部门访问级别的角色的 ARN

此机制允许您通过 IAM 角色将 IdP 的授权模型映射到特定的 AWS HealthImaging 权限。

## 2. 创建和配置 Lambda 授权器函数

创建一个 Lambda 函数，该函数将验证 JWT 令牌，并根据令牌声明评估返回相应的 IAM 角色 ARN。此函数由运行状况映像服务调用，并传递了一个包含 HealthImaging 数据存储 ID、 DICOMWeb 操作和在 HTTP 请求中找到的访问令牌的事件：

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

Lambda 授权方函数必须返回具有以下结构的 JSON 响应：

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

```
}
```

您可以参考实现示例以了解更多信息。

> **Note**
>
> 由于只有在 lambda 授权者验证访问令牌后才会回复 DICOMWeb 请求，因此必须尽可能快地执行此函数，以提供最佳 DICOMWeb API 响应时间。

要授权 HealthImaging 服务调用 lambda 授权器函数，它必须具有允许 HealthImaging 服务调用该函数的资源策略。此资源策略可以在 lambda 配置选项卡的权限菜单中创建，也可以使用 AWS CLI：

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

此资源策略允许 HealthImaging 服务在对 API 请求进行身份验证 DICOMWeb 时调用您的 Lambda 授权方。

> **Note**
>
> 稍后可以更新 lambda 资源策略，条件与ArnLike特定数据存储的 ARN 相匹配。HealthImaging

以下是 lambda 资源策略的示例：

JSON

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

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

## 3. 使用 OIDC 身份验证创建新的数据存储

要启用 OIDC 身份验证，必须使用参数为 “ AWS CLI ” 的新数据存储库。lambda-authorizer-arn如果不联系 Support，则无法在现有数据存储上启用 OIDC 身份验证。 AWS

以下是如何创建启用 OIDC 身份验证的新数据存储的示例：

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

您可以使用 AWS CLI get-datastore 命令检查特定数据存储是否启用了 OIDC 身份验证功能，并验证属性 “” 是否存在：lambdaAuthorizerArn

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

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

```
        "updatedAt": "2025-09-30T14:16:04.015000-05:00"
    }
}
```

> **Note**
>
> AWS CLI 数据存储库创建命令的执行角色必须具有相应的权限才能调用 Lambda 授权器函数。这可以缓解权限升级攻击，在这种攻击中，恶意用户可以通过数据存储授权器配置执行未经授权的 Lambda 函数。

## 异常代码

如果身份验证失败， HealthImaging 则返回以下 HTTP 错误响应代码和正文消息：

| 条件 | AHI 回应 |
| --- | --- |
| Lambda 授权器不存在或无效 | 424 授权器配置错误 |
| 由于执行失败，授权方终止 | 424 授权器失败 |
| 任何其他未映射的授权方错误 | 424 授权器失败 |
| 授权方返回的响应无效/格式不正确 | 424 授权器配置错误 |
| Authorizer 跑了 1 秒以上 | 408 授权器超时 |
| 令牌已过期或因其他原因无效 | 403 令牌无效或已过期 |
| 由于授权方配置错误，AHI 无法联合返回的 IAM 角色 | 424 授权器配置错误 |
| 授权方返回了一个空角色 | 403 访问被拒绝 |
| 返回的角色不可调用（假设角色/信任错误配置） | 424 授权器配置错误 |
| 请求速率超过 DICOMweb 网关限制 | 429 请求太多 |

| 条件 | AHI 回应 |
| --- | --- |
| 跨区域数据存储、返回角色或授权者 Account/Cross | 424 授权者跨区域访问 Account/Cross |

## 实现示例

此 Python 示例演示了一个 lambda 授权器函数，该函数验证 HealthImaging 来自事件的 C AWS ognito 访问令牌并返回具有适当权限的 IAM 角色 ARN。 DICOMWeb

Lambda 授权方实现了两种缓存机制，以减少外部调用和响应延迟。JWKS（JSON Web Key Set）每小时提取一次，并存储在函数的临时文件夹中，允许后续函数调用在本地读取它，而不是从公共网络获取。您还会注意到 token_cache 字典对象是在此 Lambda 函数的全局上下文中实例化的。所有重用相同预热 Lambda 上下文的调用都共享全局变量。因此，成功验证的令牌可以存储在此字典中，并在下次执行相同的 Lambda 函数时快速查找。缓存方法代表了一种通用的方法，可以适合大多数身份提供商发放的访问令牌。有关 AWS Cognito 特定的缓存选项，请参阅 Cognito AWS 文档的管理用户池部分和缓存部分。

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

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

# Global token cache with TTL
token_cache = {}

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

# Load environment variables once
```

```
USER_POOL_ID = os.environ['USER_POOL_ID']
CLIENT_ID = os.environ['CLIENT_ID']
ROLE_ARN = os.environ.get('AHIDICOMWEB_READONLY_ROLE_ARN', '')

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

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

    return None

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

def fetch_jwks(jwks_url):
    """Fetch JWKS from URL and cache it"""
    logger.debug('Fetching JWKS from URL')
    jwks = requests.get(jwks_url, timeout=10).json()
    # Convert to dict for faster lookups
```

```
    jwks['keys_by_kid'] = {key['kid']: key for key in jwks['keys']}
    cache_jwks(jwks)
    return jwks

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

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

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

    return cached

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

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

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

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

```
            }

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

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

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

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

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

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

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

        # Verify and decode the token (includes expiry validation)
        payload = jwt.decode(
            token,
            public_key,
            algorithms=['RS256'],
            audience=CLIENT_ID,
```

```
            issuer=f'https://cognito-idp.{region}.amazonaws.com/{USER_POOL_ID}'
        )

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

        # Cache the validated token
        cache_token(token, payload)

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

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

# 使用 AWS 导入成像数据 HealthImaging

导入是将您的医学影像数据从 Amazon S3 输入存储桶移动到 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。

> **重要提示**
>
> HealthImaging 导入作业处理 DICOM 实例二进制.dcm文件（文件）并将其转换为图像集。使用 HealthImaging 云原生操作 (APIs) 来管理数据存储和图像集。使用 HealthImaging DICOMweb 服务的表示形式来返回 DICOMweb 响应。

以下主题介绍如何使用 AWS 管理控制台、 AWS CLI和将医学影像 HealthImaging 数据导入数据存储 AWS SDKs。

主题

- 了解导入任务
- 启动导入任务
- 获取导入任务属性
- 列出导入作业

## 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 管理控制台 AWS CLI、和 AWS SDK 来启动、描述和列出导入任务。

当您将 DICOM P10 数据导入 AWS HealthImaging 数据存储时，该服务会尝试根据元数据元素根据研究 UID、系列 UID、实例 UID 的 DICOM 层次结构自动组织实例。如果导入数据的元数据元素与数据存储中的现有主影像集不冲突，则导入的数据将成为主映像。如果新导入的 DICOM P10 数据的元数据元素与现有的主影像集冲突，则新数据将被添加到非主影像集中。当数据导入创建非主映像集时，AWS 会 HealthImaging 发出一个带有 EventBridge 的事件`isPrimary: False`，写入的记录也`success.ndjson`将包含`isPrimary: False`在对象中`importResponse`。

导入数据时， HealthImaging 会执行以下操作：

- 如果在一个导入任务中导入了构成 DICOM 系列的实例，并且这些实例与数据存储中已有的实例不冲突，则所有实例都将组织成一个主映像集。
- 如果构成 DICOM 系列的实例是在两个或多个导入任务中导入的，并且这些实例与数据存储中已有的实例不冲突，则所有实例都将组织为一个主影像集。
- 如果多次导入实例，则最新版本将覆盖存储在主映像集中的任何旧版本，并且主映像集的版本号将递增。

您可以按照更新映像集元数据中所述的步骤更新主实例中的实例。

在导入过程中，私有标签（VR 类型 OB、OD、OF、OL、OV、OV、OW、UN）中大小超过 1MB 的二进制值与元数据分开存储。使用`GetDICOMInstanceMetadata`或检索这些实例的元数据时`GetDICOMSeriesMetadata`，这些较大的二进制值将被替换 BulkDataURIs，并且可以使用 `GetDICOMBulkdata` API 检索实际的二进制数据。

HealthImaging 尝试导入您的所有医学影像数据。如果在导入过程中遇到数据不一致或无法识别的数据元素，则会在`warning.ndjson`文件中为仍可能导入的 DICOM 实例 HealthImaging 添加警告。有关警告代码的完整列表，请参阅HealthImaging 警告码。

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

- 与 DICOM 系列对应的实例将自动组合成一个影像集，表示为主影像。
- 您可以在一个导入任务或多个导入任务中导入 DICOM P10 数据，该服务会将实例组织成与 DICOM 系列对应的主映像集
- 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/",
        "warningsOutputS3Uri": "s3://medical-imaging-
output/job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/WARNING/",
        "numberOfScannedFiles": 5,
        "numberOfImportedFiles": 3,
        "numberOfFilesWithCustomerError": 2,
        "numberOfFilesWithServerError": 0,
        "numberOfGeneratedImageSets": 2,
        "imageSetsSummary": [{
"imageSetId": "12345612345612345678907890789012",
                "numberOfMatchedSOPInstances": 2
            },
            {
"imageSetId": "123456123456123456789178917890 12",
                "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", "isPrimary": True}}
{"inputFile":"dicomInputFolder/1.3.51.5145.5142.20010109.1105630.1.0.1.dcm","importResponse
{"imageSetId":"12345612345612345678917891789012", "isPrimary": True}}
```

- 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"}}
```

- 该WARNING文件夹包含所有成功导入但带有警告的映像文件的结果的文件。warning.ndjson以下示例显示从 warning.ndjson 文件的输出。

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

- 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

使用StartDICOMImportJob操作开始像素数据验证检查并将数据批量导入 AWS HealthImaging 数据存储。导入任务导入 DICOM P10 文件或使用 JSON 元数据增强现有的 DICOM 文件。该inputS3Uri参数指定包含源文件的 Amazon S3 输入存储桶。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**
>
> 在开始导入任务之前，请记住以下几点：

- HealthImaging 支持使用不同的传输语法导入 DICOM P10 文件。有些文件在导入过程中保留其原始传输语法编码，而另一些文件则默认转码为 HTJ2K 无损或 JPEG 2000 Lossless，具体取决于您的数据存储配置。有关更多信息，请参阅 支持的传输语法。
- 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。
- HealthImaging 在导入过程中将长度约束应用于特定 DICOM 元素。有关更多信息，请参阅 DICOM 元素限制。
- 要导入带有 JSON 元数据覆盖的 DICOM 文件，请为`importConfiguration`参数提供将 DICOM 文件映射到其相应的 JSON 元数据文件的参数。`DicomMetadataMapping`有关更多信息，请参阅 AWS HealthImaging API 参考*`StartDICOMImportJob`*中的。

以下菜单提供了操作步骤 AWS 管理控制台 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 适用于 C++ 的 AWS SDK API 参考StartDICOMImportJob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

启动 dicom 导入作业

以下 `start-dicom-import-job` 代码示例启动 dicom 导入作业。

```
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StartDICOMImportJob中的。

Java

适用于 Java 的 SDK 2.x

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

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

```
            System.exit(1);
        }

        return "";
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考StartDICOMImportJob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 { 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 适用于 JavaScript 的 AWS SDK API 参考StartDICOMImportJob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

适用于 Python 的 SDK（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，请参阅适用StartDICOMImportJob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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅适用StartDICOMImportJob于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

## 获取导入任务属性

使用`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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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 SDKs

### C++

SDK for C++

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

```
    return outcome;
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考中的 Get DICOMImport Job。

> **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 导入作业的属性

以下 `get-dicom-import-job` 代码示例获取 dicom 导入作业的属性。

```
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》中的 Ge t DICOMImport J ob。

Java

适用于 Java 的 SDK 2.x

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

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

       return null;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考中的 Get DICOMImport J ob。

> **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: '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 适用于 JavaScript 的 AWS SDK API 参考中的 Get DICOMImport J ob。

> **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

适用于 Python 的 SDK（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 Python 版AWS SDK 中获取DICOMImport任务 (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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

```
    ENDTRY.
```

- 有关 API 的详细信息，请参阅在 SAP 的 AWS SDK 中获取 DICOMImport J ob ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 列出导入作业

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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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 SDKs

### CLI

#### AWS CLI

##### 列出 dicom 导入作业

以下 `list-dicom-import-jobs` 代码示例列出 dicom 导入作业。

```
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》中的 “列出DICOMImport作业”。

### Java

#### 适用于 Java 的 SDK 2.x

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

        try {
```

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

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

- 有关 API 的详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 “列出DICOMImport作业”。

> **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);

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

  return jobSummaries;
};
```

- 有关 API 的详细信息，请参阅《适用于 JavaScript 的 AWS SDK API 参考》中的 “列出DICOMImport作业”。

> **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

适用于 Python 的 SDK（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 版AWS SDK 中列出DICOMImport作业 (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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅在 SAP 的 AWS SDK 中列出DICOMImport作业 ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 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）。

> **重要提示**
>
> 在导入 HealthImaging 过程中，处理 DICOM 实例二进制.dcm文件（文件）并将其转换为图像集。使用 HealthImaging 云原生操作 (APIs) 来管理数据存储和图像集。使用 HealthImaging DICOMweb 服务的表示形式来返回 DICOMweb 响应。

以下主题说明了如何在 AWS 管理控制台 AWS CLI、和中使用 HealthImaging 云原生操作 AWS SDKs 来搜索影像集并获取其关联的属性、元数据和图像框。

主题


- 了解图像集
- 搜索影像集
- 获取影像集属性
- 获取影像集元数据
- 获取影像集像素数据


## 了解图像集

图像集类似 AWS 于 DICOM 系列，是 AWS 的基础。 HealthImaging图像集是在您将 DICOM 数据导入到 HealthImaging中时创建的。该服务尝试根据研究、系列和实例的 DICOM 层次结构组织导入的 P10 数据。

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）。
- 提供一种持久存储和协调重复和不一致数据的机制。导入的 P10 数据如果与存储区中已有的主影像集冲突，则将作为非主影像集保存。解决元数据冲突后，可以将该数据设为主数据。
- 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。

以下菜单更详细地描述了影像集，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，它定义了一种抽象的分组机制，用于优化与 DICOM 系列非常相似的相关医学成像数据。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。

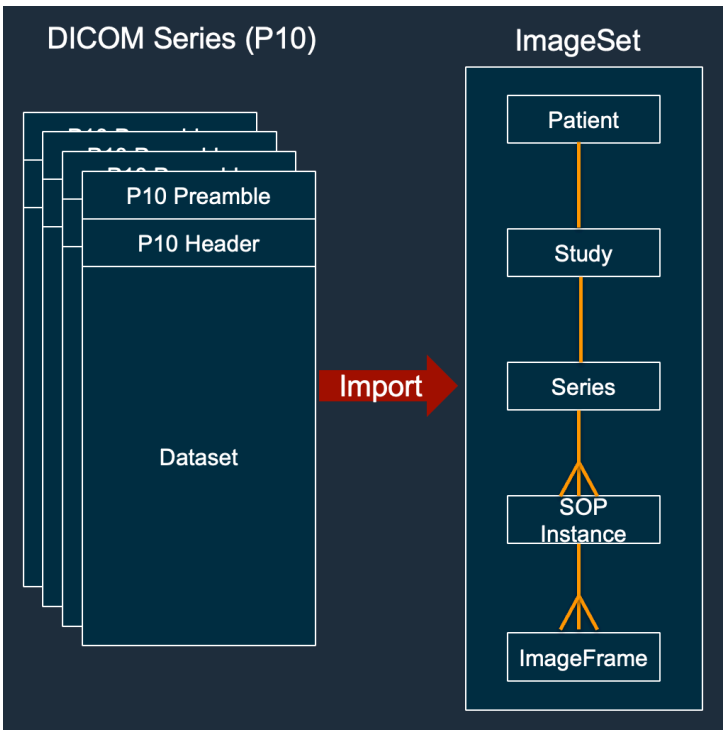
> **Note**
>
> 图像集元数据已标准化。换句话说，一组常见的属性和值映射到 DI COM 数据元素注册表中列出的患者、研究和系列级别的元素。 HealthImaging 将传入的 DICOM P10 对象分组为图像集时使用以下 DICOM 元素。
>
> 用于创建影像集的 DICOM 元素

| 元素名称 | 元素标签 |
| --- | --- |
| 学习级别的元素 | |
| Study Date | (0008,0020) |
| Accession Number | (0008,0050) |
| Patient ID | (0010,0020) |
| Study Instance UID | (0020,000D) |
| Study ID | (0020,0010) |
| 系列关卡元素 | |
| Series Instance UID | (0020,000E) |
| Series Number | (0020,0011) |

在导入过程中，某些图像集保留其原始传输语法编码，而另一些图像集则默认无损转码为 High-Throughput JPEG 2000 (HTJ2K)。如果图像集采用 HTJ2K 编码，则必须在查看之前对其进行解码。有关更多信息，请参阅支持的传输语法和图像帧解码库。
图像帧（像素数据）以 High-Throughput JPEG 2000 (HTJ2K) 编码，在观看之前必须进行解码。

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 Service-Object 配对 (SOP) 实例的 DICOM 元数据和图像帧的影像集。




Note

DICOM 导入任务：

- 请务必创建新的影像集或增加现有影像集的版本。
- 请勿删除 SOP 实例存储的重复数据。每次导入同一 SOP 实例都会使用额外的存储空间作为新的非主影像集或现有主影像集的增量版本。

- 自动将具有一致、无冲突元数据的 SOP 实例组织为主影像集，其中包含具有一致患者、研究和系列元数据元素的实例。
  - 如果在两个或多个导入任务中导入构成 DICOM 系列的实例，并且这些实例与数据存储中已有的实例不冲突，则所有实例都将组织在一个主映像集中。
- 创建包含与数据存储中已存在的主影像集冲突的 DICOM P10 数据的非主影像集。
- 将最近收到的数据保留为主影像集的最新版本。
  - 如果构成 DICOM 系列的实例是主影像集，并且再次导入了一个实例，则新副本将插入到主影像集中，版本将递增。

## 影像集元数据是什么样子？

使用`GetImageSetMetadata`操作检索影像集元数据。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。

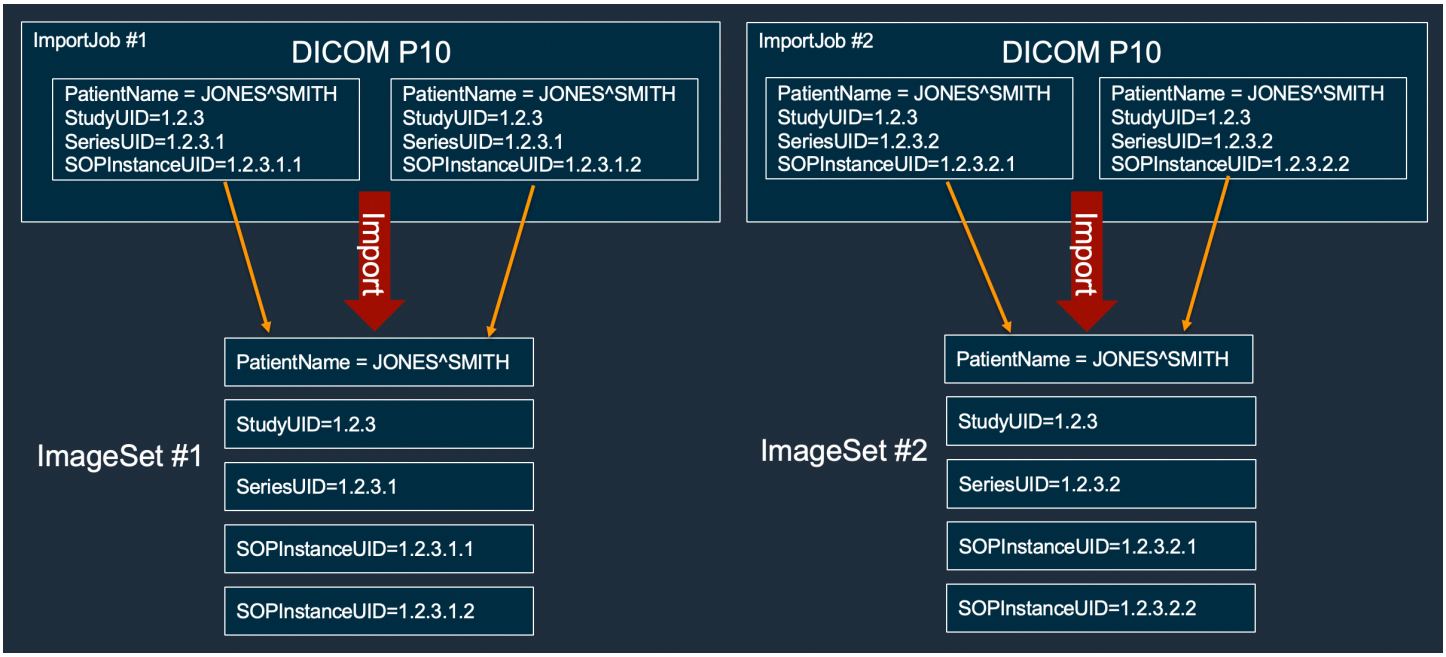
以下示例显示了 JSON 格式的图像集元数据的结构。

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

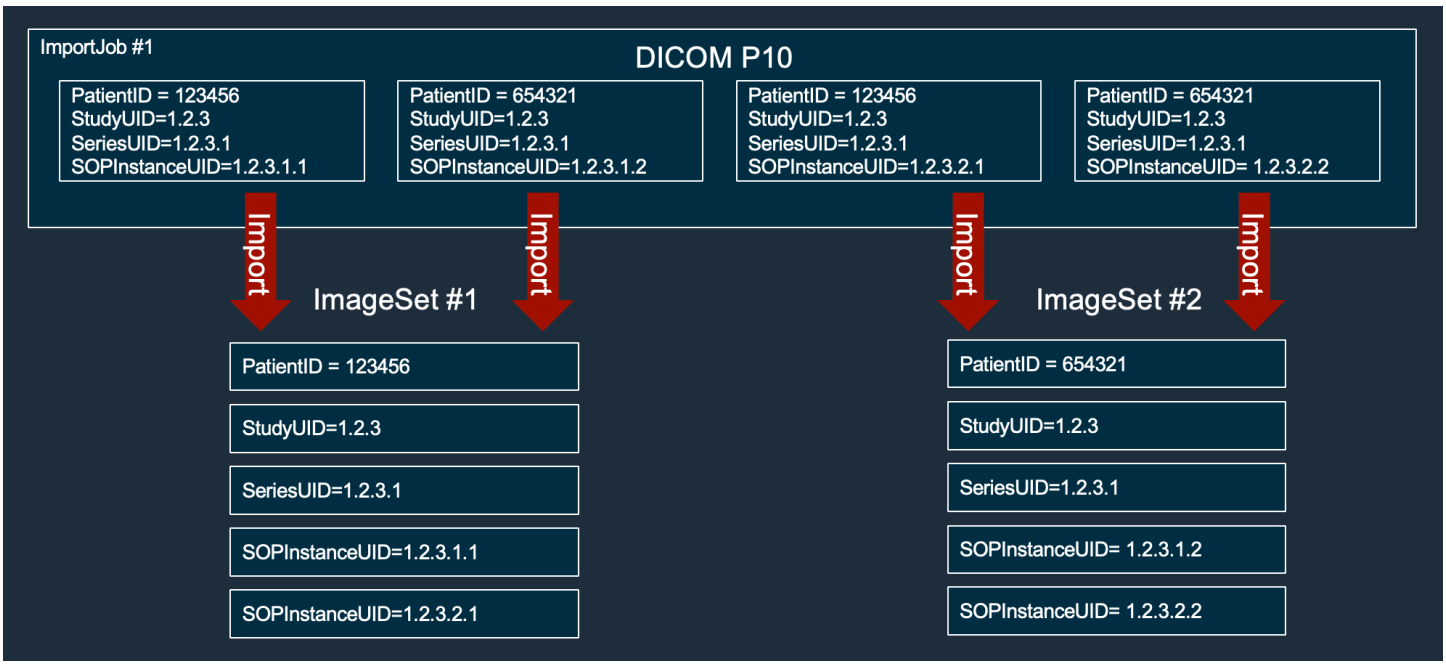
## 图像集创建示例：多个导入任务

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。



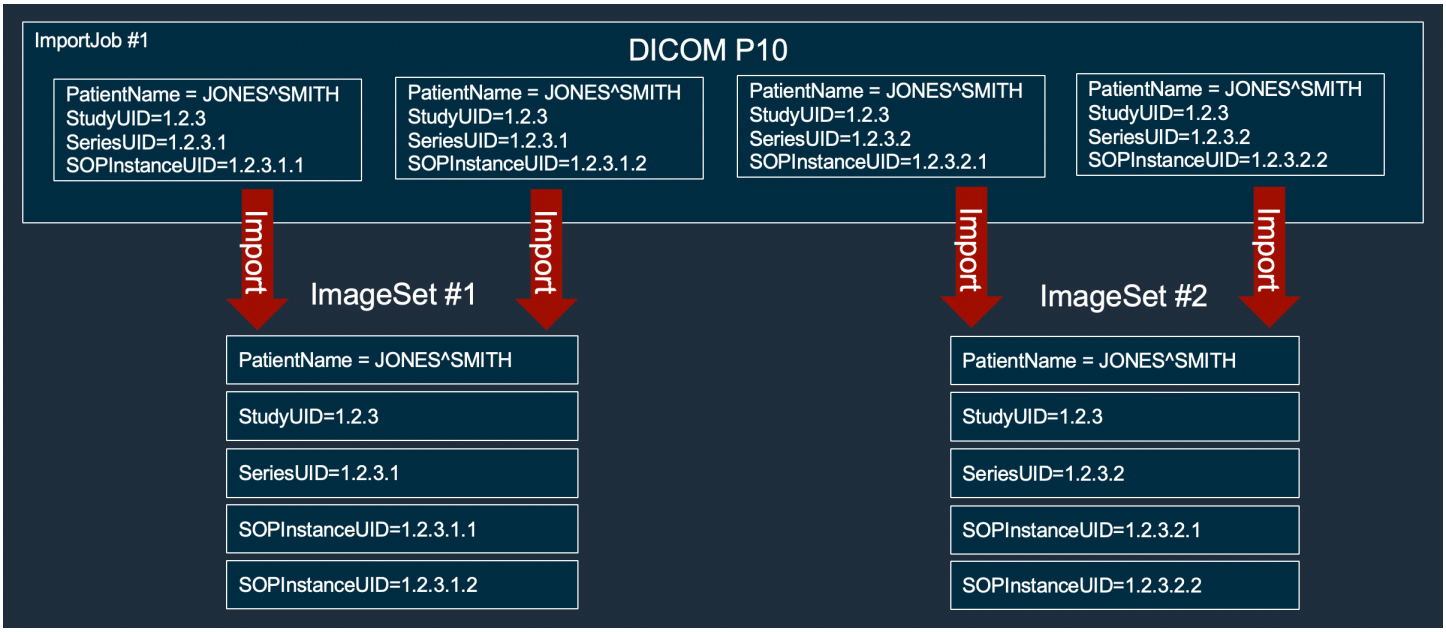

## 图像集创建示例：具有两个变体的单个导入任务

以下示例显示了一个无法合并到单个影像集的导入作业，因为实例 1 和 3 的患者 ID 与实例 2 和 4 不同。要解决此问题，您可以使用`UpdateImageSetMetadata`操作来解决患者 ID 与现有主影像集的冲突。冲突解决后，您可以使用带参数的`CopyImageSet`操作将影像集`--promoteToPrimary`添加到主影像集。



### 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

使用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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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排序。
> - 如果您使用客户拥有的 AWS KMS 密钥创建数据存储，则必须在与影像集交互之前更新 AWS KMS 密钥策略。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 唯一标识符注册表 () UIDs 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。系列实例的开头或结尾不允许出现周期 UIDs。不允许使用字母和空格，因此在复制和粘贴 UIDs时要小心。
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 UIDs ，位于所选日期范围内的系列实例将按最新顺序返回。

### 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 SDKs

### 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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
         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: DICOMSeries InstanceUID 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 适用于 C++ 的 AWS SDK 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

示例 1：使用 EQUAL 运算符搜索图像集

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

```
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 DICOMStudy日期和 DICOMStudy时间使用 BETWEEN 运算符搜索影像集

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 研究的图像集。

注意：DICOMStudy时间是可选的。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 运算符搜索图像集（之前保留了时间研究）

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

注意：采用示例中的格式（"1985-04-12T23:20:50.52Z"）提供 createdAt。

```
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 DICOMSeries instanceUID 上使用等号运算符搜索图像集，在 updateDat 上使用 BETWEEN 运算符搜索图像集，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 DICOMSeries instanceUID 上使用等号运算符搜索影像集，在 updateDat 上使用 BETWEEN 运算符搜索图像集，并在 updateDat 字段上按照 ASC 顺序对响应进行排序。

注意：采用示例中的格式（"1985-04-12T23:20:50.52Z"）提供 updatedAt。

```
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 的 SDK 2.x

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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
        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: DICOMSeries InstanceUID 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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
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: DICOMSeries InstanceUID 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
        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: DICOMSeries InstanceUID 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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        oo_result = lo_mig->searchimagesets(
          iv_datastoreid = iv_datastore_id
          io_searchcriteria = io_search_criteria ).
        DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).
        DATA(lv_count) = lines( lt_imagesets ).
        MESSAGE |Found { lv_count } image sets.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
```

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

- 有关 API 的详细信息，请参阅适用SearchImageSets于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

## 获取影像集属性

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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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。
> 使用GetDICOMInstance DICOMweb 服务的表示形式返回 DICOM 实例二进制.dcm文件（文件）。 HealthImaging有关更多信息，请参阅 从中获取 DICOM 实例 HealthImaging。

获取图像集属性

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 SDKs

### CLI

#### AWS CLI

##### 获取图像集属性

以下 `get-image-set` 代码示例可获取图像集的属性。

```
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 的 SDK 2.x

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

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

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

       return null;
   }
```

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

```
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

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

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

- 有关 API 的详细信息，请参阅适用GetImageSet于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 获取影像集元数据

使用GetImageSetMetadata操作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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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。
> 如果导入后仍在处理大型影像集元数据，则`ConflictException`可能会返回 409。处理完成后几秒钟后重试请求。
> 使用`GetDICOMInstanceMetadata` DICOMweb服务的表示形式返回 DICOM 实例元数据（`.json`文件）。 HealthImaging有关更多信息，请参阅 从中获取 DICOM 实例元数据 HealthImaging。

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 SDKs

### 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 适用于 C++ 的 AWS SDK 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

示例 1：获取没有版本的图像集元数据

以下 `get-image-set-metadata` 代码示例获取未指定版本的图像集的元数据。

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"
}
```

示例 2：获取带有版本的图像集元数据

以下 `get-image-set-metadata` 代码示例获取指定版本的图像集的元数据。

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 的 SDK 2.x

```
    public static void getMedicalImageSetMetadata(MedicalImagingClient
 medicalImagingClient,
```

```
            String destinationPath,
            String datastoreId,
            String imagesetId,
            String versionId) {

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

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


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

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

- 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 "node:fs";

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

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

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

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

  return response;
```

```
};
```

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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    " iv_version_id = '1' (optional)
    IF iv_version_id IS NOT INITIAL.
      oo_result = lo_mig->getimagesetmetadata(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id
        iv_versionid = iv_version_id ).
    ELSE.
      oo_result = lo_mig->getimagesetmetadata(
        iv_datastoreid = iv_datastore_id
        iv_imagesetid = iv_image_set_id ).
    ENDIF.
    DATA(lv_metadata_blob) = oo_result->get_imagesetmetadatablob( ).
    MESSAGE 'Image set metadata retrieved.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- 有关 API 的详细信息，请参阅适用GetImageSetMetadata于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

传输语法元数据

导入 DICOM 数据时，在影像集元数据中 HealthImaging 保留传输语法属性的原始值。导入的原始 DICOM 数据的传输语法存储为`TransferSyntaxUID`。 HealthImaging 用于表示`StoredTransferSyntaxUID`用于在数据存储中对图像帧数据进行编码的格式：`1.2.840.10008.1.2.4.202`适用于启用 HTJ2 K 的数据存储（默认）和`1.2.840.10008.1.2.4.90`启用 JPEG 2000 Lossless 的数据存储。

# 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。使用`GetImageFrame`操作检索中给定图像的 HTJ2 K 编码或原生 JPEG 2000 无损图像帧。 HealthImaging以下菜单提供了 AWS CLI 和的代码示例 AWS SDKs。有关更多信息，请参阅 AWS HealthImaging API 参考*`GetImageFrame`*中的。

> **Note**
>
> 使用`GetImageFrame`操作时，请记住以下几点：
>
> - 在导入过程中， HealthImaging 保留某些传输语法的编码，并将其他语法转码为 HTJ2 K 无损（默认）或 JPEG 2000 Lossless。该`GetImageFrame`操作以实例的存储传输语法返回图像框架。检索期间不进行任何转码，以确保检索延迟最小。在图像查看器中查看图像之前，可能需要对图像帧进行解码，具体取决于传输语法。有关更多信息，请参阅支持的传输语法和图像帧解码库。
> - 对于存储在一个或多个 HealthImaging 以 MPEG 系列传输语法（包括 MPEG-4 AVC/H.264 and HEVC/H .265）中编码的图像帧的实例 MPEG2，该`GetImageFrame`操作将以存储的传输语法返回视频对象。
> - 图像帧的传输语法在`Content-Type` HTTP标题响应元素中指定。例如，以 HTJ2 K 编码的图像帧将具有`Content-Type: image/jph header`。有关更多信息，请参阅 AWS HealthImaging API 参考*`GetImageFrame`*中的。
> - 您还可以使用`GetDICOMInstanceFrames` DICOMweb 服务的表示形式为 DICOMweb兼容的查看器和应用程序检索 DICOM 实例帧（`multipart`请求）。 HealthImaging有关更多信息，请参阅 从中获取 DICOM 实例帧 HealthImaging。

获取图像集像素数据

根据您对 AWS 的访问偏好选择菜单 HealthImaging。

## AWS 控制台

> **Note**
>
> 由于 AWS 管理控制台未提供影像查看器，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 图像帧解码库。

## AWS CLI 和 SDKs

C++

SDK for C++

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

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

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

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

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

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

    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK 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

获取图像集像素数据

以下 `get-image-frame` 代码示例获取图像帧。

```
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。有关解码和查看图像帧的信息，请参阅 HTJ2 K 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 的 SDK 2.x

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

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

.imageFrameInformation(ImageFrameInformation.builder()

.imageFrameId(imageFrameId)
                                                       .build())
                                       .build();

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_frame_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->getimageframe(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id
      io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(
        iv_imageframeid = iv_image_frame_id ) ).
    DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).
    MESSAGE 'Image frame retrieved.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image frame not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- 有关 API 的详细信息，请参阅适用GetImageFrame于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 使用 AWS 修改图像集 HealthImaging

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

> **重要提示**
>
> 在导入 HealthImaging 过程中，处理 DICOM 实例二进制.dcm文件（文件）并将其转换为图像集。使用 HealthImaging 云原生操作 (APIs) 来管理数据存储和图像集。使用 HealthImaging DICOMweb 服务的表示形式来返回 DICOMweb 响应。

HealthImaging 提供了多个云原生 APIs 以简化图像集修改过程。以下主题介绍如何使用 AWS CLI 和修改影像集 AWS SDKs。

主题


- 列出影像集版本
- 更新影像集元数据
- 复制图像集
- 删除一个影像集


## 列出影像集版本

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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。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 SDKs

### CLI

#### AWS CLI

列出图像集版本

以下 `list-image-set-versions` 代码示例列出了图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 的 SDK 2.x

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

```
                    .datastoreId(datastoreId)
                    .imageSetId(imagesetId)
                    .build();

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

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

        return null;
    }
```

- 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,
  );

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

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

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

- 有关 API 的详细信息，请参阅适用ListImageSetVersions于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

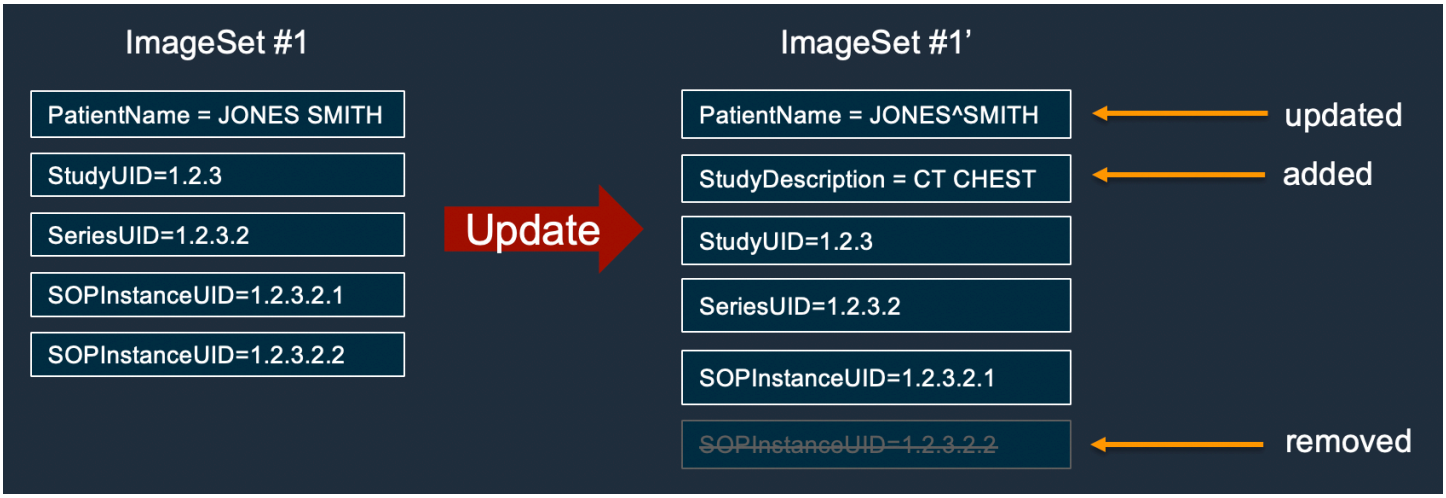
## 更新影像集元数据

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。使用该`UpdateImageSetMetadata`操作，您可以移除单个 SOP 实例，以使影像集与外部系统保持同步，并取消对图像集元数据的识别。有关更多信息，请参阅 AWS HealthImaging API 参考*`UpdateImageSetMetadata`*中的。

> **Note**
>
> Real-world 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：
>
> - 传入`--include-study-image-sets`标志以更新所有与请求的影像集共享相同研究实例 UID 的主影像集。这是一个原子操作，所有受影响的影像集的版本都将递增。注意：`revertToVersionId`操作不支持该`--include-study-image-sets`标志，因为还原会恢复之前的版本并且不应用属性更改。
> - 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 列出影像集版本。要恢复到以前的影像集版本 ID，请使用可选`revertToVersionId`参数。

- 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正在更新的影像集的相应状态和状态。您不能对LOCKED影像集执行其他写入操作。
- 如果UpdateImageSetMetadata操作不成功，请调用并查看message响应元素进行查看common errors。
- DICOM 元素约束适用于元数据更新。force请求参数允许您在需要覆盖DICOM 元数据限制的情况下更新非主影像集的元素。
- UpdateImageSet 不支持 — force 更新主影像集的 StudyInstance UID、 SeriesInstance UID 和 sopInstanceUID。
- 设置force请求参数以强制完成对非主影像集的UpdateImageSetMetadata操作。设置此参数允许对影像集进行以下更新：
  - 更新Tag.StudyInstanceUID、Tag.SeriesInstanceUIDTag.SOPInstanceUID、和Tag.StudyID属性
  - 添加、移除或更新实例级私有 DICOM 数据元素
- 要从图像集中移除实例，必须为UpdateImageSetMetadata请求提供--force参数。
- 将影像集升级为主影像集的操作将更改影像集 ID。
- 更新 VR=SQ 属性时，将更新整个序列属性。此 API 不支持部分序列属性更新。

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

#### 示例 1：在图像集元数据中插入或更新属性

以下 update-image-set-metadata 示例在图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 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"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"
}
```

#### 示例 2：从图像集元数据中删除属性

以下 update-image-set-metadata 示例从图像集元数据中删除属性。

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

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "removableAttributes": "{\"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"
}
```

示例 3：从图像集元数据中删除实例

以下 update-image-set-metadata 示例从图像集元数据中删除实例。

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

metadata-updates.json 的内容

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

输出：

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

示例 4：将图像集恢复到以前的版本

以下update-image-set-metadata示例说明如何将图像集恢复到以前的版本。CopyImageSet 和 UpdateImageSetMetadata 操作会创建图像集的新版本。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
    --latest-version-id 3 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates '{"revertToVersionId": "1"}'
```

输出：

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

```
    "updatedAt": 1680042257.908
}
```

示例 5：向实例添加私有 DICOM 数据元素

以下 update-image-set-metadata 示例演示如何将私有元素添加到图像集中的指定实例。DICOM 标准允许将私有数据元素用于通信标准数据元素中无法包含的信息。您可以通过 UpdateImageSetMetadata 操作创建、更新和删除私有数据元素。

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

metadata-updates.json 的内容

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

输出：

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

示例 6：更新实例的私有 DICOM 数据元素

以下 update-image-set-metadata 示例演示如何更新属于图像集内某个实例的私有数据元素的值。

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

metadata-updates.json 的内容

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

输出：

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

示例 7：使用 force 参数更新 SOPInstanceUID

以下 update-image-set-metadata 示例演示如何通过使用 force 参数覆盖 DICOM 元数据约束来更新 SOPInstanceUID。

```
aws medical-imaging update-image-set-metadata \
        --datastore-id 12345678901234567890123456789012 \
        --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
```

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

metadata-updates.json 的内容

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

输出：

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
    "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 的 SDK 2.x

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

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

           UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

使用案例 #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, force);
```

使用案例 #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, force);
```

使用案例 #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, force);
```

使用案例 #4：恢复到以前的版本。

```
                // In this case, revert to previous version.
                String revertVersionId =
Integer.toString(Integer.parseInt(versionid) - 1);
```

```
                MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                        .revertToVersionId(revertVersionId)
                        .build();
                updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
 imagesetId,
                        versionid, metadataRemoveUpdates, force);
```

- 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.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
```

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

使用案例 #1：插入或更新属性并强制更新。

```
    const insertAttributes = JSON.stringify({
      SchemaVersion: 1.1,
      Study: {
        DICOM: {
          StudyDescription: "CT CHEST",
        },
      },
    });
```

```
const updateMetadata = {
  DICOMUpdates: {
    updatableAttributes: new TextEncoder().encode(insertAttributes),
  },
};

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

使用案例 #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,
);
```

使用案例 #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,
);
```

使用案例 #4：恢复到早期版本。

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

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

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（Boto3）

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


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

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

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

以下代码实例化对象。 MedicalImagingWrapper

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

使用案例 #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, force
            )
```

使用案例 #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, force
            )
```

使用案例 #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, force
            )
```

使用案例 #4：恢复到早期版本。

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

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

- 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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_latest_version_id = '1'
        " iv_force = abap_false
        oo_result = lo_mig->updateimagesetmetadata(
          iv_datastoreid = iv_datastore_id
          iv_imagesetid = iv_image_set_id
          iv_latestversionid = iv_latest_version_id
          io_updateimagesetmetupdates = io_metadata_updates
          iv_force = iv_force ).
        DATA(lv_new_version) = oo_result->get_latestversionid( ).
        MESSAGE |Image set metadata updated to version: { lv_new_version }.| TYPE
'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migservicequotaexcdex.
        MESSAGE 'Service quota exceeded.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
```

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

- 有关 API 的详细信息，请参阅适用UpdateImageSetMetadata于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

您可以使用、和 DeleteImageSet API 在影像集之间移动 SOP 实例、解决元数据元素冲突以及向主影像集添加或移除实例。CopyImageSet UpdateImageSetMetadata

您可以通过DeleteImageSet操作将图像集从主收藏中移除。

## 更新主影像集的元数据

1. 使用 CopyImageSet 操作创建非主影像集，该影像集是您要修改的主影像集的副本。假设它以非主影像集 ID 的103785414bc2c89330f7ce51bbd13f7a形式返回。

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

2. 使用 UpdateImageSetMetadata 操作对非主影像集(103785414bc2c89330f7ce51bbd13f7a)进行更改。例如，更改 patienTid。

```
aws medical-imaging update-image-set-metadata \
```

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

3. 删除您正在修改的主影像集。

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

4. 使用带有参数的 CopyImageSet 操作将更新的图像集`--promoteToPrimary`添加到主集合中。

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

5. 删除非主影像集。

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

## 将非主映像集设为主映像

1. 使用 UpdateImageSetMetadata 操作来解决与现有主影像集的冲突。

```
aws medical-imaging update-image-set-metadata \
    --region us-west-2 \
    --datastore-id a8d19e7875e1532d9b5652f6b25e12c9 \
    --image-set-id 103785414bc2c89330f7ce51bbd13f7a \
    --latest-version-id 1 \
```

```
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates '{
    "DICOMUpdates": {
      "updatableAttributes": "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":
      {\"PatientID\":\"1234\"}}}"
    }
  }'
```

2. 冲突解决后，使用带有参数的 CopyImageSet 操作将影像集`--promoteToPrimary`添加到主影像集合中。

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

3. 确认 CopyImageSet 操作成功后，删除源非主影像集。

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

# 复制图像集

使用`CopyImageSet`操作复制中设置的图像 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。您可以复制到新的影像集中，以分割影像集，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。

> **Note**
>
> 使用`CopyImageSet`操作时，请记住以下几点：
>
> - 该`CopyImageSet`操作将创建新的影像集或新版本的`destinationImageSet`。有关更多信息，请参阅 列出影像集版本。
> - 复制是一个异步过程。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要求 SOP 实例 UIDs 在图片集中是唯一的。
- 您可以使用复制 SOP 实例copiableAttributes的子集。这允许您从中挑选一个或多个 SOP 实例sourceImageSet，以复制到destinationImageSet。
- 如果CopyImageSet操作不成功，请致电GetImageSet并查看message酒店。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。除非提供了可选force参数，否则该CopyImageSet操作需要sourceImageSet并destinationImageSet具有一致的元数据。
- 即使和之间存在不一致的元数据元素，也要将force参数设置为强制执行该sourceImageSet操作destinationImageSet。在这些情况下，患者、研究和系列元数据在中保持不变destinationImageSet。

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 SDKs

CLI

AWS CLI

示例 1：复制没有目标的图像集

以下 copy-image-set 示例制作没有目标的图像集的副本。

```
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"
}
```

示例 2：复制带有目标的图像集

以下 copy-image-set 示例制作带有目标的图像集的副本。

```
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"
}
```

示例 3：将实例子集从源图像集复制到目标图像集

以下 copy-image-set 示例将一个 DICOM 实例从源图像集复制到目标图像集。提供 force 参数是为了覆盖“患者”、“研究”和“系列”级别属性中的不一致。

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet":
 {"latestVersionId": "1","DICOMCopies": {"copiableAttributes":
 "{\"SchemaVersion\":\"1.1\",\"Study\":{\"Series\":
{\"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3666.0\":
{\"Instances\":
{\"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3669.0\":
{}}}}}}"}},"destinationImageSet": {"imageSetId":
 "b9eb50d8ee682eb9fcf4acbf92f62bb7","latestVersionId": "1"}}' \
    --force
```

输出：

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042505.135,
        "imageSetId": "b9eb50d8ee682eb9fcf4acbf92f62bb7",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042505.135,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
```

```
}
```

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 的 SDK 2.x

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

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

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

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

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

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

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

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

用于创建可复制的属性的实用程序函数。

```
   /**
    * Create a JSON string of copiable image instances.
    *
    * @param imageSetId - The image set ID.
    * @param subsets    - The subsets to copy.
    * @return A JSON string of copiable image instances.
    */
   private static String getCopiableAttributesJSON(String imageSetId,
Vector<String> subsets) {
       StringBuilder subsetInstanceToCopy = new StringBuilder(
               """
                       {
                         "SchemaVersion": 1.1,
                         "Study": {
                           "Series": {
                               "
                                """
       );

       subsetInstanceToCopy.append(imageSetId);

       subsetInstanceToCopy.append(
               """
                               ": {
                               "Instances": {
                       """
       );

       for (String subset : subsets) {
           subsetInstanceToCopy.append('"' + subset + "\": {},");
       }
       subsetInstanceToCopy.deleteCharAt(subsetInstanceToCopy.length() - 1);
       subsetInstanceToCopy.append("""
                        }
                      }
                   }
                 }
               }
               """);
       return subsetInstanceToCopy.toString();
   }
```

- 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.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = [],
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
```

```
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
    force: force,
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

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

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

    params.copyImageSetInformation.dicomCopies = copySubsetsJson;
  }

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

```
    //      },
    //       datastoreId: 'xxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
    //             latestVersionId: '4',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //      }
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

复制没有目标的映像集。

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

复制带有目标的映像集。

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

使用目标复制映像集的子集并强制复制。

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

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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,
        force=False,
        subsets=[],
    ):
        """
        Copy an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param destination_image_set_id: The ID of the optional destination image
set.
        :param destination_version_id: The ID of the optional destination image
set version.
        :param force: Force the copy.
        :param subsets: The optional subsets to copy. For example:
["12345678901234567890123456789012"].
        :return: The copied image set ID.
        """
        try:
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }
            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
                    "imageSetId": destination_image_set_id,
                    "latestVersionId": destination_version_id,
                }
            if len(subsets) > 0:
                copySubsetsJson = {
                    "SchemaVersion": "1.1",
                    "Study": {"Series": {"imageSetId": {"Instances": {}}}},
                }
```

```
                for subset in subsets:
                    copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[
                        subset
                    ] = {}

                copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
                    "copiableAttributes": json.dumps(copySubsetsJson)
                }
            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
                force=force,
            )
        except ClientError as err:
            logger.error(
                "Couldn't copy image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return copy_results["destinationImageSetProperties"]["imageSetId"]
```

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,
                force=force,
            )
```

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,
                force=force,
            )
```

复制映像集的子集。

```
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }

            if len(subsets) > 0:
                copySubsetsJson = {
                    "SchemaVersion": "1.1",
                    "Study": {"Series": {"imageSetId": {"Instances": {}}}},
                }

                for subset in subsets:
                    copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[
                        subset
                    ] = {}

                copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
                    "copiableAttributes": json.dumps(copySubsetsJson)
                }

            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
```

```
                copyImageSetInformation=copy_image_set_information,
                force=force,
            )
```

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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_version_id = '1'
        " iv_destination_image_set_id =
'12345678901234567890123456789012345678901234567890' (optional)
        " iv_destination_version_id = '1' (optional)
        " iv_force = abap_false
        DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
          iv_latestversionid = iv_source_version_id ).
        DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(
          io_sourceimageset = lo_source_info ).
        IF iv_destination_image_set_id IS NOT INITIAL AND
           iv_destination_version_id IS NOT INITIAL.
          DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
            iv_imagesetid = iv_destination_image_set_id
            iv_latestversionid = iv_destination_version_id ).
          lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
            io_sourceimageset = lo_source_info
```

```
            io_destinationimageset = lo_dest_info ).
        ENDIF.
        oo_result = lo_mig->copyimageset(
          iv_datastoreid = iv_datastore_id
          iv_sourceimagesetid = iv_source_image_set_id
          io_copyimagesetinformation = lo_copy_info
          iv_force = iv_force ).
        DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
        DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
        MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migservicequotaexcdex.
        MESSAGE 'Service quota exceeded.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用CopyImageSet于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 删除一个影像集

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 管理控制台 和 AWS CLI 和的代码示例 AWS SDKs。有关更多信息，请参阅 AWS HealthImaging API 参考*DeleteImageSet*中的。

删除图像集

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 SDKs

C++

SDK for C++

```
//! Routine which deletes an AWS HealthImaging image set.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The image set ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::deleteImageSet(
        const Aws::String &dataStoreID, const Aws::String &imageSetID,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::DeleteImageSetRequest request;
    request.SetDatastoreId(dataStoreID);
```

```
    request.SetImageSetId(imageSetID);
    Aws::MedicalImaging::Model::DeleteImageSetOutcome outcome =
 client.DeleteImageSet(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted image set " << imageSetID
                  << " from data store " << dataStoreID << std::endl;
    }
    else {
        std::cerr << "Error deleting image set " << imageSetID << " from data
 store "
                  << dataStoreID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK 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

#### 删除图像集

以下 `delete-image-set` 代码示例删除图像集。

```
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 的 SDK 2.x

```
   public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imagesetId) {
       try {
           DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId)
                   .build();

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

           System.out.println("The image set was deleted.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->deleteimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
    MESSAGE 'Image set deleted.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- 有关 API 的详细信息，请参阅适用DeleteImageSet于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 使用 AWS 为资源添加标签 HealthImaging

您可以以标签的形式将元数据分配给 HealthImaging 资源（数据存储和图像集）。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。标签的用途并非用于私人或敏感数据。

以下主题介绍如何使用 AWS 管理控制台 AWS CLI、和 AWS SDKs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
- 列出资源的标签
- 取消标记资源


## 标记资源

使用`TagResource`操作来标记 AWS 中的数据存储和图像集 HealthImaging。以下代码示例描述了如何将`TagResource`操作与 AWS 管理控制台 AWS CLI、和一起使用 AWS SDKs。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

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 SDKs

### CLI

#### AWS CLI

##### 示例 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：标记图像集

以下 `tag-resource` 代码示例可标记图像集。

```
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 的 SDK 2.x

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

           medicalImagingClient.tagResource(tagResourceRequest);

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

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
TRY.
```

```
        " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
        lo_mig->tagresource(
          iv_resourcearn = iv_resource_arn
          it_tags = it_tags ).
        MESSAGE 'Resource tagged successfully.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用TagResource于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

## 列出资源的标签

使用ListTagsForResource操作列出 AWS 中数据存储和图像集的标签 HealthImaging。以下代码示例描述了如何将ListTagsForResource操作与 AWS 管理控制台 AWS CLI、和一起使用 AWS SDKs。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 SDKs

### CLI

#### AWS CLI

**示例 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：列出图像集的资源标签**

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

```
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 的 SDK 2.x

```
   public static ListTagsForResourceResponse
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
           String resourceArn) {
       try {
           ListTagsForResourceRequest listTagsForResourceRequest =
ListTagsForResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .build();

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

       return null;
   }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
        oo_result = lo_mig->listtagsforresource( iv_resourcearn =
 iv_resource_arn ).
        DATA(lt_tags) = oo_result->get_tags( ).
        DATA(lv_count) = lines( lt_tags ).
        MESSAGE |Found { lv_count } tags for resource.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用ListTagsForResource于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 取消标记资源

使用`UntagResource`操作可取消标记 AWS HealthImaging 中的数据存储和图像集。以下代码示例描述了如何将`UntagResource`操作与 AWS 管理控制台 AWS CLI、和一起使用 AWS SDKs。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

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 SDKs

### CLI

#### AWS CLI

##### 示例 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：取消标记图像集

以下 `untag-resource` 代码示例可取消标记图像集。

```
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 的 SDK 2.x

```
    public static void untagMedicalImagingResource(MedicalImagingClient
 medicalImagingClient,
            String resourceArn,
            Collection<String> tagKeys) {
```

```
        try {
            UntagResourceRequest untagResourceRequest =
 UntagResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .tagKeys(tagKeys)
                    .build();

            medicalImagingClient.untagResource(untagResourceRequest);

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

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
        lo_mig->untagresource(
          iv_resourcearn = iv_resource_arn
          it_tagkeys = it_tag_keys ).
        MESSAGE 'Resource untagged successfully.' TYPE 'I'.
```

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

- 有关 API 的详细信息，请参阅适用UntagResource于 S AP 的AWS SDK ABAP 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。

> **示例可用性**
>
> 找不到所需的内容？ 使用本页右侧边栏上的 “提供反馈” 链接请求代码示例。

# 使用的代码示 HealthImaging 例 AWS SDKs

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的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合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，请参阅将此服务与 AWS SDK。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。

代码示例


- 使用的基本示 HealthImaging 例 AWS SDKs
  - 你好 HealthImaging
  - HealthImaging 使用的操作 AWS SDKs
    - 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 使用场景 AWS SDKs
    - 使用 AWS SDK 开始使用 HealthImaging 图像集和图像框架
    - 使用 SDK 标记 HealthImaging 数据存储 AWS
    - 使用 SDK 为 HealthImaging 图像集添加标签 AWS


# 使用的基本示 HealthImaging 例 AWS SDKs

以下代码示例说明如何使用 with 的基础 AWS HealthImaging 知识 AWS SDKs。

示例


- 你好 HealthImaging
- HealthImaging 使用的操作 AWS SDKs
    - 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

## 你好 HealthImaging

以下代码示例展示了如何开始使用 HealthImaging。

C++

SDK for C++

CMakeLists.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 适用于 C++ 的 AWS SDK 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（Boto3）

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

logger = logging.getLogger(__name__)


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

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

```
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise


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

- 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，请参阅将此服务与 AWS SDK。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 使用的操作 AWS SDKs

以下代码示例演示了如何使用执行单个 HealthImaging操作 AWS SDKs。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。

这些摘录调用 HealthImaging API，是大型程序的代码摘录，这些程序必须在上下文中运行。您可以在HealthImaging 使用场景 AWS SDKs中结合上下文查看操作。

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

示例 1：复制没有目标的图像集

以下 `copy-image-set` 示例制作没有目标的图像集的副本。

```
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"
}
```

示例 2：复制带有目标的图像集

以下 copy-image-set 示例制作带有目标的图像集的副本。

```
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"
}
```

示例 3：将实例子集从源图像集复制到目标图像集

以下 copy-image-set 示例将一个 DICOM 实例从源图像集复制到目标图像集。提供 force 参数是为了覆盖“患者”、“研究”和“系列”级别属性中的不一致。

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet":
 {"latestVersionId": "1","DICOMCopies": {"copiableAttributes":
 "{\"SchemaVersion\":\"1.1\",\"Study\":{\"Series\":
{\"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3666.0\":
{\"Instances\":
{\"1.3.6.1.4.1.5962.99.1.3673257865.2104868982.1369432891697.3669.0\":
{}}}}}}"}},"destinationImageSet": {"imageSetId":
 "b9eb50d8ee682eb9fcf4acbf92f62bb7","latestVersionId": "1"}}' \
    --force
```

输出：

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042505.135,
        "imageSetId": "b9eb50d8ee682eb9fcf4acbf92f62bb7",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042505.135,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
```

```
}
```

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 的 SDK 2.x

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

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

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

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

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

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

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

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

用于创建可复制的属性的实用程序函数。

```
   /**
    * Create a JSON string of copiable image instances.
    *
    * @param imageSetId - The image set ID.
    * @param subsets    - The subsets to copy.
    * @return A JSON string of copiable image instances.
    */
   private static String getCopiableAttributesJSON(String imageSetId,
Vector<String> subsets) {
       StringBuilder subsetInstanceToCopy = new StringBuilder(
               """
                       {
                         "SchemaVersion": 1.1,
                         "Study": {
                           "Series": {
                               "
                                """
       );

       subsetInstanceToCopy.append(imageSetId);

       subsetInstanceToCopy.append(
               """
                               ": {
                               "Instances": {
                       """
       );

       for (String subset : subsets) {
           subsetInstanceToCopy.append('"' + subset + "\": {},");
       }
       subsetInstanceToCopy.deleteCharAt(subsetInstanceToCopy.length() - 1);
       subsetInstanceToCopy.append("""
                        }
                      }
                   }
                 }
               }
               """);
       return subsetInstanceToCopy.toString();
   }
```

- 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.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = [],
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
```

```
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
    force: force,
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

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

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

    params.copyImageSetInformation.dicomCopies = copySubsetsJson;
  }

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

```
    //      },
    //       datastoreId: 'xxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
    //             latestVersionId: '4',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //      }
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

复制没有目标的映像集。

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

复制带有目标的映像集。

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

使用目标复制映像集的子集并强制复制。

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

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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,
        force=False,
        subsets=[],
    ):
        """
        Copy an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param destination_image_set_id: The ID of the optional destination image
set.
        :param destination_version_id: The ID of the optional destination image
set version.
        :param force: Force the copy.
        :param subsets: The optional subsets to copy. For example:
["12345678901234567890123456789012"].
        :return: The copied image set ID.
        """
        try:
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }
            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
                    "imageSetId": destination_image_set_id,
                    "latestVersionId": destination_version_id,
                }
            if len(subsets) > 0:
                copySubsetsJson = {
                    "SchemaVersion": "1.1",
                    "Study": {"Series": {"imageSetId": {"Instances": {}}}},
                }
```

```
                for subset in subsets:
                    copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[
                        subset
                    ] = {}

                copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
                    "copiableAttributes": json.dumps(copySubsetsJson)
                }
            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
                force=force,
            )
        except ClientError as err:
            logger.error(
                "Couldn't copy image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return copy_results["destinationImageSetProperties"]["imageSetId"]
```

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,
                force=force,
            )
```

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,
                force=force,
            )
```

复制映像集的子集。

```
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }

            if len(subsets) > 0:
                copySubsetsJson = {
                    "SchemaVersion": "1.1",
                    "Study": {"Series": {"imageSetId": {"Instances": {}}}},
                }

                for subset in subsets:
                    copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[
                        subset
                    ] = {}

                copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
                    "copiableAttributes": json.dumps(copySubsetsJson)
                }

            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
```

```
                copyImageSetInformation=copy_image_set_information,
                force=force,
            )
```

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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_source_version_id = '1'
        " iv_destination_image_set_id =
'12345678901234567890123456789012345678901234567890' (optional)
        " iv_destination_version_id = '1' (optional)
        " iv_force = abap_false
        DATA(lo_source_info) = NEW /aws1/cl_migcpsrcimagesetinf00(
          iv_latestversionid = iv_source_version_id ).
        DATA(lo_copy_info) = NEW /aws1/cl_migcpimagesetinfmtion(
          io_sourceimageset = lo_source_info ).
        IF iv_destination_image_set_id IS NOT INITIAL AND
           iv_destination_version_id IS NOT INITIAL.
          DATA(lo_dest_info) = NEW /aws1/cl_migcopydstimageset(
            iv_imagesetid = iv_destination_image_set_id
            iv_latestversionid = iv_destination_version_id ).
          lo_copy_info = NEW /aws1/cl_migcpimagesetinfmtion(
            io_sourceimageset = lo_source_info
```

```
            io_destinationimageset = lo_dest_info ).
        ENDIF.
        oo_result = lo_mig->copyimageset(
          iv_datastoreid = iv_datastore_id
          iv_sourceimagesetid = iv_source_image_set_id
          io_copyimagesetinformation = lo_copy_info
          iv_force = iv_force ).
        DATA(lo_dest_props) = oo_result->get_dstimagesetproperties( ).
        DATA(lv_new_id) = lo_dest_props->get_imagesetid( ).
        MESSAGE |Image set copied with new ID: { lv_new_id }.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migservicequotaexcdex.
        MESSAGE 'Service quota exceeded.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用CopyImageSet于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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

示例 1：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。在未指定 a 的情况下创建数据存储时`--lossless-storage-format`，AWS HealthImaging 默认为 HTJ2K（高吞吐量 JPEG 2000）。

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

输出：

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

示例 2：采用 JPEG 2000 无损存储格式创建数据存储

使用 JPEG 2000 无损存储格式配置的数据存储将会转码并以 JPEG 2000 格式保存无损图像帧。然后，无需转码即可在 JPEG 2000 无损存储中检索图像帧。以下 `create-datastore` 代码示例创建配置为 JPEG 2000 无损存储格式的数据存储，其名称为 `my-datastore`。

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

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 的 SDK 2.x

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

       return "";
   }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅适用CreateDatastore于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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 的 SDK 2.x

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

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅适用DeleteDatastore于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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 适用于 C++ 的 AWS SDK 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

删除图像集

以下 delete-image-set 代码示例删除图像集。

```
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 的 SDK 2.x

```
   public static void deleteMedicalImageSet(MedicalImagingClient
medicalImagingClient,
           String datastoreId,
           String imagesetId) {
       try {
           DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
                   .datastoreId(datastoreId)
                   .imageSetId(imagesetId)
                   .build();

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

           System.out.println("The image set was deleted.");
       } catch (MedicalImagingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

### SAP ABAP

#### 适用于 SAP ABAP 的 SDK

```
TRY.
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
    oo_result = lo_mig->deleteimageset(
      iv_datastoreid = iv_datastore_id
      iv_imagesetid = iv_image_set_id ).
    MESSAGE 'Image set deleted.' TYPE 'I'.
  CATCH /aws1/cx_migaccessdeniedex.
    MESSAGE 'Access denied.' TYPE 'I'.
  CATCH /aws1/cx_migconflictexception.
    MESSAGE 'Conflict error.' TYPE 'I'.
  CATCH /aws1/cx_miginternalserverex.
    MESSAGE 'Internal server error.' TYPE 'I'.
  CATCH /aws1/cx_migresourcenotfoundex.
    MESSAGE 'Image set not found.' TYPE 'I'.
  CATCH /aws1/cx_migthrottlingex.
    MESSAGE 'Request throttled.' TYPE 'I'.
  CATCH /aws1/cx_migvalidationex.
    MESSAGE 'Validation error.' TYPE 'I'.
ENDTRY.
```

- 有关 API 的详细信息，请参阅适用DeleteImageSet于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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 适用于 C++ 的 AWS SDK API 参考中的 Get DICOMImport Job。

> **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 导入作业的属性

以下 `get-dicom-import-job` 代码示例获取 dicom 导入作业的属性。

```
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》中的 Ge t DICOMImport J ob。

Java

适用于 Java 的 SDK 2.x

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

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

       return null;
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考中的 Get DICOMImport J ob。

> **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: '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 适用于 JavaScript 的 AWS SDK API 参考中的 Get DICOMImport J ob。

> **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

适用于 Python 的 SDK（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 Python 版AWS SDK 中获取DICOMImport任务 (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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

```
    ENDTRY.
```

- 有关 API 的详细信息，请参阅在 SAP 的 AWS SDK 中获取 DICOMImport J ob ABAP 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，请参阅将此服务与 AWS SDK。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

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

示例 2：获取为配置的数据存储属性 JPEG2000

以下 get-datastore 代码示例获取配置为 JPEG 2000 无损存储格式的数据存储的属性。

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

输出：

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

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 的 SDK 2.x

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

       return null;
   }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅适用GetDatastore于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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 适用于 C++ 的 AWS SDK 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

获取图像集像素数据

以下 `get-image-frame` 代码示例获取图像帧。

```
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。有关解码和查看图像帧的信息，请参阅 HTJ2 K 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 的 SDK 2.x

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

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

.imageFrameInformation(ImageFrameInformation.builder()

.imageFrameId(imageFrameId)
                                                                .build())
                                                .build();

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_frame_id = '12345678901234567890123456789012345678901234567890'
        oo_result = lo_mig->getimageframe(
          iv_datastoreid = iv_datastore_id
          iv_imagesetid = iv_image_set_id
          io_imageframeinformation = NEW /aws1/cl_migimageframeinfmtion(
            iv_imageframeid = iv_image_frame_id ) ).
        DATA(lv_frame_blob) = oo_result->get_imageframeblob( ).
        MESSAGE 'Image frame retrieved.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
```

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

- 有关 API 的详细信息，请参阅适用GetImageFrame于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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` 代码示例可获取图像集的属性。

```
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 的 SDK 2.x

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

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

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

```
        return null;
    }
```

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

```
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

### 适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_version_id = '1' (optional)
        IF iv_version_id IS NOT INITIAL.
          oo_result = lo_mig->getimageset(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id
            iv_versionid = iv_version_id ).
        ELSE.
          oo_result = lo_mig->getimageset(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id ).
        ENDIF.
        DATA(lv_state) = oo_result->get_imagesetstate( ).
        MESSAGE |Image set retrieved with state: { lv_state }.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Image set not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用GetImageSet于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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 适用于 C++ 的 AWS SDK 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

示例 1：获取没有版本的图像集元数据

以下 `get-image-set-metadata` 代码示例获取未指定版本的图像集的元数据。

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"
}
```

示例 2：获取带有版本的图像集元数据

以下 `get-image-set-metadata` 代码示例获取指定版本的图像集的元数据。

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 的 SDK 2.x

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

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

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


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

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

```
            System.exit(1);
        }
    }
```

- 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 "node:fs";

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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
        " iv_version_id = '1' (optional)
        IF iv_version_id IS NOT INITIAL.
          oo_result = lo_mig->getimagesetmetadata(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id
            iv_versionid = iv_version_id ).
        ELSE.
          oo_result = lo_mig->getimagesetmetadata(
            iv_datastoreid = iv_datastore_id
            iv_imagesetid = iv_image_set_id ).
        ENDIF.
        DATA(lv_metadata_blob) = oo_result->get_imagesetmetadatablob( ).
        MESSAGE 'Image set metadata retrieved.' TYPE 'I'.
```

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

- 有关 API 的详细信息，请参阅适用GetImageSetMetadata于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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 导入作业。

```
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》中的 “列出DICOMImport作业”。

Java

适用于 Java 的 SDK 2.x

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

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

        return new ArrayList<>();
```

```
    }
```

- 有关 API 的详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 “列出DICOMImport作业”。

> **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);

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

```
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/
dicom_import',
  //             datastoreId: '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《适用于 JavaScript 的 AWS SDK API 参考》中的 “列出DICOMImport作业”。

> **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

适用于 Python 的 SDK（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 版AWS SDK 中列出DICOMImport作业 (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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅在 SAP 的 AWS SDK 中列出DICOMImport作业 ABAP 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，请参阅将此服务与 AWS SDK。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 的 SDK 2.x

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

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

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

```
        return null;
    }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅适用ListDatastores于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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` 代码示例列出了图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 的 SDK 2.x

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

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

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

       return null;
   }
```

- 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,
  );

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

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

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

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

- 有关 API 的详细信息，请参阅适用ListImageSetVersions于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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：列出图像集的资源标签

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

```
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 的 SDK 2.x

```
   public static ListTagsForResourceResponse
listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
           String resourceArn) {
       try {
           ListTagsForResourceRequest listTagsForResourceRequest =
ListTagsForResourceRequest.builder()
                   .resourceArn(resourceArn)
                   .build();

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

       return null;
   }
```

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

SAP ABAP

适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
        oo_result = lo_mig->listtagsforresource( iv_resourcearn =
 iv_resource_arn ).
        DATA(lt_tags) = oo_result->get_tags( ).
        DATA(lv_count) = lines( lt_tags ).
        MESSAGE |Found { lv_count } tags for resource.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用ListTagsForResource于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
         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: DICOMSeries InstanceUID 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 适用于 C++ 的 AWS SDK 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

示例 1：使用 EQUAL 运算符搜索图像集

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

```
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 DICOMStudy日期和 DICOMStudy时间使用 BETWEEN 运算符搜索影像集

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 研究的图像集。

注意：DICOMStudy时间是可选的。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 运算符搜索图像集（之前保留了时间研究）

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

注意：采用示例中的格式（"1985-04-12T23:20:50.52Z"）提供 createdAt。

```
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 DICOMSeries instanceUID 上使用等号运算符搜索图像集，在 updateDat 上使用 BETWEEN 运算符搜索图像集，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 DICOMSeries instanceUID 上使用等号运算符搜索影像集，在 updateDat 上使用 BETWEEN 运算符搜索图像集，并在 updateDat 字段上按照 ASC 顺序对响应进行排序。

注意：采用示例中的格式（"1985-04-12T23:20:50.52Z"）提供 updatedAt。

```
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 的 SDK 2.x

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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
        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: DICOMSeries InstanceUID 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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
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: DICOMSeries InstanceUID 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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: 使用 DICOMStudy日期和 DICOMStudy时间的 BETWEEN 运算符。

```
        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: DICOMSeries InstanceUID 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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        oo_result = lo_mig->searchimagesets(
          iv_datastoreid = iv_datastore_id
          io_searchcriteria = io_search_criteria ).
        DATA(lt_imagesets) = oo_result->get_imagesetsmetadatasums( ).
        DATA(lv_count) = lines( lt_imagesets ).
```

```
        MESSAGE |Found { lv_count } image sets.| TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_migconflictexception.
        MESSAGE 'Conflict error.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用SearchImageSets于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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《适用于 C++ 的 AWS SDK API 参考》中的 “启动 DICOMImport Job”。

> **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 导入作业

以下 `start-dicom-import-job` 代码示例启动 dicom 导入作业。

```
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》中的 “启动 DICOMImport J ob”。

Java

适用于 Java 的 SDK 2.x

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

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

       return "";
   }
```

- 有关 API 的详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 “启动 DICOMImport Job”。

> **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《适用于 JavaScript 的 AWS SDK API 参考》中的 “启动 DICOMImport Job”。

> **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

适用于 Python 的 SDK（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 版AWS SDK 中启动DICOMImport作业 (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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
TRY.
    " iv_job_name = 'import-job-1'
    " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
    " iv_role_arn = 'arn:aws:iam::123456789012:role/ImportJobRole'
```

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

- 有关 API 的详细信息，请参阅在 SAP 的 AWS SDK 中启动 DICOMImport J ob ABAP 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，请参阅将此服务与 AWS SDK。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：标记图像集

以下 tag-resource 代码示例可标记图像集。

```
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 的 SDK 2.x

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

           medicalImagingClient.tagResource(tagResourceRequest);

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

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
TRY.
```

```
        " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
        lo_mig->tagresource(
          iv_resourcearn = iv_resource_arn
          it_tags = it_tags ).
        MESSAGE 'Resource tagged successfully.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
        MESSAGE 'Access denied.' TYPE 'I'.
      CATCH /aws1/cx_miginternalserverex.
        MESSAGE 'Internal server error.' TYPE 'I'.
      CATCH /aws1/cx_migresourcenotfoundex.
        MESSAGE 'Resource not found.' TYPE 'I'.
      CATCH /aws1/cx_migthrottlingex.
        MESSAGE 'Request throttled.' TYPE 'I'.
      CATCH /aws1/cx_migvalidationex.
        MESSAGE 'Validation error.' TYPE 'I'.
    ENDTRY.
```

- 有关 API 的详细信息，请参阅适用TagResource于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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：取消标记图像集

以下 untag-resource 代码示例可取消标记图像集。

```
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 的 SDK 2.x

```
    public static void untagMedicalImagingResource(MedicalImagingClient
 medicalImagingClient,
            String resourceArn,
            Collection<String> tagKeys) {
        try {
```

```
            UntagResourceRequest untagResourceRequest =
 UntagResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .tagKeys(tagKeys)
                    .build();

            medicalImagingClient.untagResource(untagResourceRequest);

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

- 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 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_resource_arn = 'arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012'
        lo_mig->untagresource(
          iv_resourcearn = iv_resource_arn
          it_tagkeys = it_tag_keys ).
        MESSAGE 'Resource untagged successfully.' TYPE 'I'.
      CATCH /aws1/cx_migaccessdeniedex.
```

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

- 有关 API 的详细信息，请参阅适用UntagResource于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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

示例 1：在图像集元数据中插入或更新属性

以下 `update-image-set-metadata` 示例在图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 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"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"
}
```

示例 2：从图像集元数据中删除属性

以下 update-image-set-metadata 示例从图像集元数据中删除属性。

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

metadata-updates.json 的内容

```
{
    "DICOMUpdates": {
        "removableAttributes": "{\"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"
}
```

示例 3：从图像集元数据中删除实例

以下 update-image-set-metadata 示例从图像集元数据中删除实例。

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

metadata-updates.json 的内容

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

输出：

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

```
}
```

示例 4：将图像集恢复到以前的版本

以下update-image-set-metadata示例说明如何将图像集恢复到以前的版本。CopyImageSet 和 UpdateImageSetMetadata 操作会创建图像集的新版本。

```
aws medical-imaging update-image-set-metadata \
    --datastore-id 12345678901234567890123456789012 \
    --image-set-id 53d5fdb05ca4d46ac7ca64b06545c66e \
    --latest-version-id 3 \
    --cli-binary-format raw-in-base64-out \
    --update-image-set-metadata-updates '{"revertToVersionId": "1"}'
```

输出：

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

示例 5：向实例添加私有 DICOM 数据元素

以下 update-image-set-metadata 示例演示如何将私有元素添加到图像集中的指定实例。DICOM 标准允许将私有数据元素用于通信标准数据元素中无法包含的信息。您可以使用 UpdateImageSetMetadata 操作创建、更新和删除私有数据元素。

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

metadata-updates.json 的内容

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

输出：

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

示例 6：更新实例的私有 DICOM 数据元素

以下 update-image-set-metadata 示例演示如何更新属于图像集内某个实例的私有数据元素的值。

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

metadata-updates.json 的内容

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

```
    }
}
```

输出：

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

示例 7：使用强制参数更新 SOPInstance UID

以下update-image-set-metadata示例说明如何使用 force 参数覆盖 DICOM 元数据约束来更新 SOPInstance UID。

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

metadata-updates.json 的内容

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

输出：

```
{
    "latestVersionId": "2",
    "imageSetWorkflowStatus": "UPDATING",
    "updatedAt": 1680042257.908,
    "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",
    "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 的 SDK 2.x

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

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

            UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

使用案例 #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, force);
```

使用案例 #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, force);
```

使用案例 #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, force);
```

使用案例 #4：恢复到以前的版本。

```
            // In this case, revert to previous version.
            String revertVersionId =
Integer.toString(Integer.parseInt(versionid) - 1);
            MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                    .revertToVersionId(revertVersionId)
                    .build();
            updateMedicalImageSetMetadata(medicalImagingClient, datastoreId,
imagesetId,
                    versionid, metadataRemoveUpdates, force);
```

- 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.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
```

```
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

使用案例 #1：插入或更新属性并强制更新。

```
    const insertAttributes = JSON.stringify({
      SchemaVersion: 1.1,
      Study: {
        DICOM: {
          StudyDescription: "CT CHEST",
        },
      },
    });

    const updateMetadata = {
      DICOMUpdates: {
        updatableAttributes: new TextEncoder().encode(insertAttributes),
      },
    };

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

使用案例 #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,
);
```

使用案例 #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,
    );
```

使用案例 #4：恢复到早期版本。

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

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

- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK 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

适用于 Python 的 SDK（Boto3）

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


    def update_image_set_metadata(
```

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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param metadata: The image set metadata as a dictionary.
            For example {"DICOMUpdates": {"updatableAttributes":
            "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{\"PatientName\":
\"Garcia^Gloria\"}}}"}}
        :param: force: Force the update.
        :return: The updated image set metadata.
        """
        try:
            updated_metadata =
 self.health_imaging_client.update_image_set_metadata(
                imageSetId=image_set_id,
                datastoreId=datastore_id,
                latestVersionId=version_id,
                updateImageSetMetadataUpdates=metadata,
                force=force,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update image set metadata. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return updated_metadata
```

以下代码实例化对象。 MedicalImagingWrapper

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

使用案例 #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, force
            )
```

使用案例 #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, force
            )
```

使用案例 #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, force
            )
```

使用案例 #4：恢复到早期版本。

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

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

- 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。

## SAP ABAP

### 适用于 SAP ABAP 的 SDK

```
    TRY.
        " iv_datastore_id = '12345678901234567890123456789012345678901234567890'
        " iv_image_set_id = '12345678901234567890123456789012345678901234567890'
```

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

- 有关 API 的详细信息，请参阅适用UpdateImageSetMetadata于 S AP 的AWS SDK ABAP 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，请参阅将此服务与 AWS SDK。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 使用场景 AWS SDKs

以下代码示例向您展示了如何在中 HealthImaging实现常见场景 AWS SDKs。这些场景向您展示了如何通过在其中调用多个函数 HealthImaging 或与其他函数组合来完成特定任务 AWS 服务。每个场景都包含完整源代码的链接，您可以在其中找到有关如何设置和运行代码的说明。

场景以中等水平的经验为目标，可帮助您结合具体环境了解服务操作。

示例

- 使用 AWS SDK 开始使用 HealthImaging 图像集和图像框架
- 使用 SDK 标记 HealthImaging 数据存储 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

该实现构造为命令行应用程序。

- 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 IDs 的图像集。
- 检索影像集 IDs 的图像框。
- 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 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、解码并验证映像帧。

```
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《适用于 C++ 的 AWS SDK API Reference》中的以下主题。
  - DeleteImageSet
  - Get DICOMImport Job
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - 开始 DICOMImport Job

> **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

编排步骤（index.js）。

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
```

```
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
```

```
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
```

```
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Health Imaging Workflow",
    description:
      "Work with DICOM images using an AWS Health Imaging data store.",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes]
 [-v|--verbose]",
  });
}
```

部署资源（deploy-steps.js）。

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
```

```
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
```

```
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
```

```
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

复制 DICOM 文件（dataset-steps.js）。

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
```

```
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
```

```
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = "input/";
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

开始导入到数据存储（import-steps.js）。

```
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
```

```
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

获取图像集 IDs (image-set-steps.js-)。

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";
```

```
import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
```

```
      state.manifestContent.jobSummary.imageSetsSummary.reduce((ids, next) => {
        return Object.assign({}, ids, {
          [next.imageSetId]: next.imageSetId,
        });
      }, {});
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

获取图像框 IDs (image-frame-steps.js)。

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node:util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
```

```
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
```

```
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
```

```
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
);
```

验证映像帧（verify-steps.js）。使用AWS HealthImaging 像素数据验证库进行验证。

```
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
```

```
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
```

```
    default: true,
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (/** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid}
 and sop ${sopInstanceUid}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(
```

```
                    `Verification tool exited with code ${code} for image set
 ${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

摧毁资源（clean-up-steps.js）。

```
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
```

```
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */
```

```
const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
```

```
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- 有关 API 详细信息，请参阅《适用于 JavaScript 的 AWS SDK API Reference》中的以下主题。
  - DeleteImageSet
  - Get DICOMImport Job
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - 开始 DICOMImport Job

> **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

适用于 Python 的 SDK（Boto3）

使用必要的资源创建 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(
            "../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml"
        ) as setup_file:
            setup_template = setup_file.read()
        print(f"\t\tCreating {stack_name}.")
        stack = self.cf_resource.create_stack(
            StackName=stack_name,
            TemplateBody=setup_template,
            Capabilities=["CAPABILITY_NAMED_IAM"],
            Parameters=[
                {
                    "ParameterKey": "datastoreName",
                    "ParameterValue": data_store_name,
                },
                {
                    "ParameterKey": "userAccountID",
                    "ParameterValue": account_id,
                },
            ],
        )
        print("\t\tWaiting for stack to deploy. This typically takes a minute or
 two.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
        waiter.wait(StackName=stack.name)
        stack.load()
        print(f"\t\tStack status: {stack.stack_status}")

        outputs_dictionary = {
            output["OutputKey"]: output["OutputValue"] for output in
 stack.outputs
        }
        self.input_bucket_name = outputs_dictionary["BucketName"]
        self.output_bucket_name = outputs_dictionary["BucketName"]
```

```
        self.role_arn = outputs_dictionary["RoleArn"]
        self.data_store_id = outputs_dictionary["DatastoreID"]
        return stack
```

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 AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def start_dicom_import_job(
        self,
        data_store_id,
        input_bucket_name,
        input_directory,
        output_bucket_name,
        output_directory,
        role_arn,
    ):
        """
```

```
        Routine which starts a HealthImaging import job.

        :param data_store_id: The HealthImaging data store ID.
        :param input_bucket_name: The name of the Amazon S3 bucket containing the
 DICOM files.
        :param input_directory: The directory in the S3 bucket containing the
 DICOM files.
        :param output_bucket_name: The name of the S3 bucket for the output.
        :param output_directory: The directory in the S3 bucket to store the
 output.
        :param role_arn: The ARN of the IAM role with permissions for the import.
        :return: The job ID of the import.
        """

        input_uri = f"s3://{input_bucket_name}/{input_directory}/"
        output_uri = f"s3://{output_bucket_name}/{output_directory}/"
        try:
            job = self.medical_imaging_client.start_dicom_import_job(
                jobName="examplejob",
                datastoreId=data_store_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_uri,
                outputS3Uri=output_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

获取由 DICOM 导入任务创建的影像集。

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""
```

```
    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
        """
        Retrieves the image sets created for an import job.

        :param datastore_id: The HealthImaging data store ID
        :param import_job_id: The import job ID
        :return: List of image set IDs
        """

        import_job = self.medical_imaging_client.get_dicom_import_job(
            datastoreId=datastore_id, jobId=import_job_id
        )

        output_uri = import_job["jobProperties"]["outputS3Uri"]

        bucket = output_uri.split("/")[2]
        key = "/".join(output_uri.split("/")[3:])

        # Try to get the manifest.
        retries = 3
        while retries > 0:
            try:
                obj = self.s3_client.get_object(
                    Bucket=bucket, Key=key + "job-output-manifest.json"
                )
                body = obj["Body"]
                break
            except ClientError as error:
                retries = retries - 1
```

```
                time.sleep(3)
        try:
            data = json.load(body)
            expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
            image_sets = expression.search(data)
        except json.decoder.JSONDecodeError as error:
            image_sets = import_job["jobProperties"]

        return image_sets


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

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

获取影像集的影像帧信息。

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_frames_for_image_set(self, datastore_id, image_set_id,
 out_directory):
        """
        Get the image frames for an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param out_directory: The directory to save the file.
        :return: The image frames.
        """
        image_frames = []
        file_name = os.path.join(out_directory,
 f"{image_set_id}_metadata.json.gzip")
        file_name = file_name.replace("/", "\\\\")
        self.get_image_set_metadata(file_name, datastore_id, image_set_id)
        try:
            with gzip.open(file_name, "rb") as f_in:
                doc = json.load(f_in)
            instances = jmespath.search("Study.Series.*.Instances[].*[]", doc)
            for instance in instances:
                rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
```

```
                rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)
                image_frames_json = jmespath.search("ImageFrames[][]", instance)
                for image_frame in image_frames_json:
                    checksum_json = jmespath.search(
                        "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",
                        image_frame,
                    )
                    image_frame_info = {
                        "imageSetId": image_set_id,
                        "imageFrameId": image_frame["ID"],
                        "rescaleIntercept": rescale_intercept,
                        "rescaleSlope": rescale_slope,
                        "minPixelValue": image_frame["MinPixelValue"],
                        "maxPixelValue": image_frame["MaxPixelValue"],
                        "fullResolutionChecksum": checksum_json["Checksum"],
                    }
                    image_frames.append(image_frame_info)
            return image_frames
        except TypeError:
            return {}
        except ClientError as err:
            logger.error(
                "Couldn't get image frames for image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        return image_frames


    def get_image_set_metadata(
        self, metadata_file, datastore_id, image_set_id, version_id=None
    ):
        """
        Get the metadata of an image set.

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """
```

```
        try:
            if version_id:
                image_set_metadata =
 self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
                image_set_metadata =
 self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
            with open(metadata_file, "wb") as f:
                for chunk in
 image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)

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

下载、解码并验证映像帧。

```
class MedicalImagingWrapper:
    """Encapsulates AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client
```

```
    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


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

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


    def download_decode_and_check_image_frames(
        self, data_store_id, image_frames, out_directory
    ):
        """
        Downloads image frames, decodes them, and uses the checksum to validate
        the decoded images.
```

```
        :param data_store_id: The HealthImaging data store ID.
        :param image_frames: A list of dicts containing image frame information.
        :param out_directory: A directory for the downloaded images.
        :return: True if the function succeeded; otherwise, False.
        """
        total_result = True
        for image_frame in image_frames:
            image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
            self.get_pixel_data(
                image_file_path,
                data_store_id,
                image_frame["imageSetId"],
                image_frame["imageFrameId"],
            )

            image_array = self.jph_image_to_opj_bitmap(image_file_path)
            crc32_checksum = image_frame["fullResolutionChecksum"]
            # Verify checksum.
            crc32_calculated = zlib.crc32(image_array)
            image_result = crc32_checksum == crc32_calculated
            print(
                f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
 {image_result }"
            )
            total_result = total_result and image_result
        return total_result

    @staticmethod
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array
```

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 AWS HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.medical_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
```

```
            return metadata_summaries


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

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

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API Reference》中的以下主题。
  - DeleteImageSet
  - Get DICOMImport Job
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - 开始 DICOMImport Job

> **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，请参阅将此服务与 AWS SDK。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 标记 HealthImaging 数据存储 AWS

以下代码示例演示如何标记 HealthImaging 数据存储。

Java

适用于 Java 的 SDK 2.x

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 Reference》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《适用于 JavaScript 的 AWS SDK API Reference》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

适用于 Python 的 SDK（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/1234567890123456789012345678901 2"

    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 Reference》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，请参阅将此服务与 AWS SDK。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 的 SDK 2.x

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 Reference》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《适用于 JavaScript 的 AWS SDK API Reference》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

适用于 Python 的 SDK（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 Reference》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，请参阅将此服务与 AWS SDK。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。

# 监控 AWS HealthImaging

监控和日志是维护 AWS 安全性、可靠性、可用性和性能的重要组成部分 HealthImaging。 AWS 提供以下日志和监控工具 HealthImaging，供您监视、报告问题并在适当时自动采取措施：

- AWS CloudTrail捕获由您的账户或代表您的 AWS 账户进行的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以识别哪些用户和帐户拨打了电话 AWS、发出呼叫的源 IP 地址以及呼叫发生的时间。有关更多信息，请参阅 AWS CloudTrail 《用户指南》。
- Amazon 会实时 CloudWatch监控您的 AWS 资源和您运行 AWS 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 CloudWatch 跟踪您的 Amazon EC2 实例的 CPU 使用率或其他指标，并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。
- Amazon EventBridge 是一项无服务器事件总线服务，可以轻松地将您的应用程序与来自各种来源的数据连接起来。 EventBridge 提供来自您自己的应用程序、 Software-as-a-Service (SaaS) 应用程序和 AWS 服务的实时数据流，并将这些数据路由到 Lambda 等目标。这使您能够监控服务中发生的事件，并构建事件驱动的架构。有关更多信息，请参阅 Amazon EventBridge 用户指南。

主题


- AWS CloudTrail 与一起使用 HealthImaging
- 将 Amazon CloudWatch 与 HealthImaging
- 将 Amazon EventBridge 与 HealthImaging


# AWS CloudTrail 与一起使用 HealthImaging

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 AWS 账户中查看、搜索和下载最新事件。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 管理控制台，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。默认情况下，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 APIs 每秒交易量 (tps) 较高的数据事件保留的，因此出于成本考虑，您可以选择不 CloudTrail 保存日志。使用 HealthImaging，AWS API 参考中列出的所有原生 HealthImaging API 操作都被归类为管理事件，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。DICOMweb WADO-RS API 操作在中被归类为数据事件 CloudTrail，因此，您必须选择加入它们。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的正在从中检索 DICOM 数据 HealthImaging和记录数据事件。

每个事件或日志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CloudTrailuserIdentity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": "XXXXXXXXXXXXXXXXXXX",
        "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"
}
```


# 将 Amazon CloudWatch 与 HealthImaging

您可以使用监控 AWS HealthImaging CloudWatch，它会收集原始数据并将其处理为可读的近乎实时的指标。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。还可以设置特定阈值监视警报，在达到对应阈值时发送通知或采取行动。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

HealthImaging 将以下类型的指标发布到`AWS/HealthImaging`命名空间 CloudWatch中：

- API 指标- HealthImaging API 操作的调用次数
- HealthImaging 指标-数据存储和账户级资源使用情况

> **Note**
>
> - 报告了大多数指标 HealthImaging APIs。
> - HealthImaging 指标仅适用于 2026 年 2 月 9 日之后或通过提交支持案例创建的数据存储。

# AWS HealthImaging API 指标

下表列出了的指标和维度 HealthImaging。每个都以用户指定数据范围的频率计数表示。

指标

| 指标 | 说明 |
| --- | --- |
| `CallCount` | 呼叫的次数 APIs。可以为账户或指定的数据存储报告此问题。<br><br>单位：计数<br><br>有效统计数据：Sum、Count<br><br>维度：操作、数据存储 ID、数据存储类型 |

您可以通过 AWS 管理控制台 AWS CLI、或 CloudWatch API 获取指标。 HealthImaging 您可以通过其中一个 Amazon AWS 软件开发套件 (SDKs) 或 CloudWatch API 工具来使用 API。 CloudWatch HealthImaging 控制台根据来自 CloudWatch API 的原始数据显示图表。

您必须具有相应的 CloudWatch 权限才能 HealthImaging 进行监控 CloudWatch。有关更多信息，请参阅《CloudWatch 用户指南》 CloudWatch中的身份和访问管理。

# AWS HealthImaging 指标

`AWS/HealthImaging`命名空间包括以下账户和数据存储级别的指标。

## 账户级别指标

账户级指标可提供您账户中所有数据存储的汇总可见性。

账户级别指标

| 指标 | 说明 |
| --- | --- |
| `DataStoreCount` | 处于活动状态的数据存储数量。<br><br>单位：计数<br><br>有效统计数据：Sum、Average |

| 指标 | 说明 |
| --- | --- |
| ImageSetCount | 所有数据存储中的影像集总数。<br><br>单位：计数<br><br>有效统计数据：Sum、Average |
| StorageBytes | 在以下存储层的所有数据存储中存储的数据量（以字节为单位）：<br><br>• 频繁访问（FrequentAccessStorage）<br>• 存档即时访问（ArchiveInstantAccessStorage）<br><br>该值的计算方法是将所有数据存储中所有影像集的大小相加。<br><br>有效的存储层筛选器（参见StorageTier 维度）：<br><br>• FrequentAccessStorage<br>• ArchiveInstantAccessStorage<br>• AllStorage<br><br>单位：字节<br><br>有效统计数据：Sum、Average |

## 数据存储级别的指标

数据存储级别的指标提供了对单个数据存储的详细可见性。

数据存储级别的指标

| 指标 | 说明 |
| --- | --- |
| TotalImageSetCount | 数据存储中的影像集总数。 |

| 指标 | 说明 |
| --- | --- |
| | 单位：计数<br><br>有效统计数据：Sum、Average |
| PrimaryImageSetCount | 数据存储中的主映像集的数量。<br><br>单位：计数<br><br>有效统计数据：Sum、Average |
| SmallImageSetCount | 数据存储中小于 5MB 的影像集数量。<br><br>单位：计数<br><br>有效统计数据：Sum、Average |
| StorageBytes | 在以下存储层的所有数据存储中存储的数据量（以字节为单位）：<br><br>• 频繁访问（FrequentAccessStorage）<br>• 存档即时访问（ArchiveInstantAccessStorage）<br><br>该值的计算方法是将数据存储中所有影像集的大小相加。<br><br>有效的存储层筛选器（参见StorageTier 维度）：<br><br>• FrequentAccessStorage<br>• ArchiveInstantAccessStorage<br>• AllStorage<br><br>单位：字节<br><br>有效统计数据：Sum、Average |

| 指标 | 说明 |
| --- | --- |
| DICOMStudyCount | 数据存储中 DICOM 研究的数量。<br>单位：计数<br>有效统计数据：Sum、Average |
| DICOMSeriesCount | 数据存储中 DICOM 系列的数量。<br>单位：计数<br>有效统计数据：Sum、Average |
| DICOMInstanceCount | 数据存储中 DICOM 实例的数量。<br>单位：计数<br>有效统计数据：Sum、Average |
| StructuredStorageBytes | 数据存储索引的结构化存储量（以字节为单位）。<br>单位：字节<br>有效统计数据：Sum、Average |

## HealthImaging 尺寸（英寸） CloudWatch

以下维度用于筛选 HealthImaging 指标。

Dimensions

| 维度 | 说明 |
| --- | --- |
| AccountId | 此维度筛选已识别 AWS 账户的数据。 |
| DatastoreId | 此维度仅筛选已识别数据存储的数据。 |
| StorageTier | 此维度按以下存储层筛选数据： |

| 维度 | 说明 |
| --- | --- |
| | • FrequentAccessStorage — 频繁访问存储中用于影像集的字节数。<br>• ArchiveInstantAccessStorage — 存档即时访问存储中用于影像集的字节数。<br>• AllStorage — 所有存储层的总字节数。 |

## 访问 HealthImaging 指标

您可以使用以下方式访问 HealthImaging 指标：

- AWS 管理控制台-在 CloudWatch 控制台中查看指标
- AWS CLI-使用 CloudWatch CLI 命令
- CloudWatch API-通过我们 CloudWatch 的 API 工具 AWS SDKs 进行访问

您必须具有相应的权限才能 HealthImaging 进行监控 CloudWatch。有关更多信息，请参阅《CloudWatch 用户指南》 CloudWatch中的身份和访问管理。

## 设置 HealthImaging 指标

要在您的 CloudWatch 账户中接收 HealthImaging 指标，您需要创建一个服务相关角色 HealthImaging 来代表您发布指标。有关如何创建服务相关角色 HealthImaging的详细信息，请参阅使用服务相关角色。

## 查看 HealthImaging 指标

查看指标（CloudWatch 控制台）

1. 登录 AWS 管理控制台 并打开CloudWatch控制台。
2. 选择 “指标”，选择 “所有指标”，然后选择**AWS/HealthImaging**。
3. 选择尺寸：
   - 按账户 ID-查看账户级别指标
   - 按数据存储 ID-查看数据存储级别的指标
4. 选择指标名称，然后选择添加到图表。

5. 为日期范围选择一个值，将显示指标数量。

## 使用创建警报 CloudWatch

CloudWatch 警报在指定时间段内监视单个指标，并执行一项或多项操作：向亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 主题或 Auto Scaling 策略发送通知。一个或多个操作基于指标在您指定的多个时间段内相对于给定阈值的值。 CloudWatch 还可以在警报状态发生变化时向您发送 Amazon SNS 消息。

CloudWatch 警报仅在状态发生变化并且持续到您指定的时间段内时才会调用操作。有关更多信息，请参阅使用 CloudWatch警报。

# 将 Amazon EventBridge 与 HealthImaging

Amazon EventBridge 是一项无服务器服务，它使用事件将应用程序组件连接在一起，使您可以更轻松地构建可扩展的事件驱动应用程序。的基础 EventBridge 是创建将事件路由到目标的规则。AWS HealthImaging 为其提供状态变更的持久交付 EventBridge。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

主题

- HealthImaging 事件已发送至 EventBridge
- HealthImaging 事件结构和示例

## HealthImaging 事件已发送至 EventBridge

下表列出了发送 EventBridge 给处理的所有 HealthImaging 事件。

| HealthImaging 事件类型 | 州 |
| --- | --- |
| 数据存储事件 | |
| 创建数据存储 | `CREATING` |
| 创建数据存储失败 | `CREATE_FAILED` |
| 数据存储已创建 | `ACTIVE` |
| 正在删除数据存储 | `DELETING` |

| HealthImaging 事件类型 | 州 |
| --- | --- |
| 数据存储已删除 | DELETED |
| 有关更多信息，请参阅 AW S HealthImaging API 参考中的 DataStoreStoreStatus。 | |
| 导入作业事件 | |
| 导入 Job 已提交 | SUBMITTED |
| 正在导入 Job | IN_PROGRESS |
| 导入 Job 已完成 | COMPLETED |
| 导入 Job 失败 | FAILED |
| 有关更多信息，请参阅 AWS HealthImaging API 参考中的 JobStat us。 | |
| 图像集事件 | |
| 图像集已创建 | CREATED |
| 影像集复制 | COPYING |
| 使用只读权限复制图像集 | COPYING_WITH_READ_ONLY_ACCESS |
| 图像集已复制 | COPIED |
| 影像集复制失败 | COPY_FAILED |
| 图像集更新 | UPDATING |
| 图像集已更新 | UPDATED |
| 图像集更新失败 | UPDATE_FAILED |
| 正在删除图像集 | DELETING |
| 图像集已删除 | DELETED |
| 有关更多信息，请参阅 AWS HealthImaging API 参考ImageSetWorkflowStatus中的。 | |

# HealthImaging 事件结构和示例

HealthImaging 事件是具有 JSON 结构的对象，其中还包含元数据详细信息。您可以使用元数据作为输入来重新创建事件或了解更多信息。所有关联的元数据字段都列在以下菜单的代码示例下的表格中。有关更多信息，请参阅 Amazon EventBridge 用户指南中的事件结构参考。

> **Note**
>
> HealthImaging 事件结构的`source`属性是`aws.medical-imaging`。

### 数据存储事件

#### Data Store Creating

##### 州-**CREATING**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Creating",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "CREATING"
    }
}
```

#### Data Store Creation Failed

##### 州-**CREATE_FAILED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
```

```
    "detail-type": "Data Store Creation Failed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "CREATE_FAILED"
    }
}
```

### Data Store Created

#### 州-**ACTIVE**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Created",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "ACTIVE"
    }
}
```

### Data Store Deleting

#### 州-**DELETING**

```
{
    "version": "0",
```

```
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Deleting",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "DELETING"
    }
}
```

## Data Store Deleted

### 州-**DELETED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Deleted",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "DELETED"
    }
}
```

数据存储事件-元数据描述

| Name | Type | 说明 |
| --- | --- | --- |
| version | 字符串 | EventBridge 事件架构版本。 |
| id | 字符串 | 为每个事件生成的版本 4 UUID。 |
| detail-type | 字符串 | 正在发送的事件的类型。 |
| source | 字符串 | 标识生成事件的服务。 |
| account | 字符串 | 数据存储所有者的 12 位数 AWS 账户 ID。 |
| time | 字符串 | 事件发生的时间。 |
| region | 字符串 | 标识数据存储的 AWS 区域。 |
| resources | 数组（字符串） | 包含数据存储的 ARN 的 JSON 数组。 |
| detail | object | 包含关于事件信息的 JSON 对象。 |
| detail.imagingVersion | 字符串 | 跟踪事件详细信息架构变更 HealthImaging的版本 ID。 |
| detail.datastoreId | 字符串 | 与状态更改事件关联的数据存储 ID。 |
| detail.datastoreName | 字符串 | 数据存储名称。 |
| detail.datastoreStatus | 字符串 | 当前的数据存储状态。 |

## 导入作业事件

### Import Job Submitted

#### 州-**SUBMITTED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Submitted",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "SUBMITTED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

### Import Job In Progress

#### 州-**IN_PROGRESS**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job In Progress",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
```

```
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "IN_PROGRESS",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

### Import Job Completed

#### 州-**COMPLETED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Completed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "COMPLETED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

### Import Job Failed

#### 州-**FAILED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Failed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
```

```
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "FAILED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

### 导入任务事件-元数据描述

| Name | Type | 说明 |
| --- | --- | --- |
| version | 字符串 | EventBridge 事件架构版本。 |
| id | 字符串 | 为每个事件生成的版本 4 UUID。 |
| detail-type | 字符串 | 正在发送的事件的类型。 |
| source | 字符串 | 标识生成事件的服务。 |
| account | 字符串 | 数据存储所有者的 12 位数 AWS 账户 ID。 |
| time | 字符串 | 事件发生的时间。 |
| region | 字符串 | 标识数据存储的 AWS 区域。 |
| resources | 数组（字符串） | 包含数据存储的 ARN 的 JSON 数组。 |
| detail | object | 包含关于事件信息的 JSON 对象。 |

| Name | Type | 说明 |
| --- | --- | --- |
| detail.imagingVersion | 字符串 | 跟踪事件详细信息架构变更 HealthImaging的版本 ID。 |
| detail.datastoreId | 字符串 | 生成状态更改事件的数据存储。 |
| detail.jobId | 字符串 | 与状态更改事件关联的导入任务 ID。 |
| detail.jobName | 字符串 | 导入任务名称。 |
| detail.jobStatus | 字符串 | 当前的工作状态。 |
| detail.inputS3Uri | 字符串 | 包含要导入的 DICOM 文件的 S3 存储桶的输入前缀路径。 |
| detail.outputS3Uri | 字符串 | 将上传 DICOM 导入任务结果的 S3 存储桶的输出前缀。 |

### 图像集事件

#### Image Set Created

##### 州-**CREATED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Created",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
```

```
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "CREATED"
   }
}
```

## Image Set Copying

### 州-**COPYING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING",
      "sourceImageSetArn": "arn:aws:medical-imaging:us-
west-2:147997158357:datastore/c381ee9b9ef34902a45b476dd7be068b/
imageset/0309de3674fd551fa7ddd2880b21f990"
   }
}
```

## Image Set Copying With Read Only Access

### 州-**COPYING_WITH_READ_ONLY_ACCESS**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying With Read Only Access",
```

```
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS"
   }
}
```

## Image Set Copied

### 州-**COPIED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copied",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPIED"
   }
}
```

### Image Set Copy Failed

#### 州-**COPY_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copy Failed",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e8882533193886 49",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPY_FAILED"
   }
}
```

### Image Set Updating

#### 州-**UPDATING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updating",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
```

```
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "UPDATING"
   }
}
```

### Image Set Updated

#### 州-**UPDATED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updated",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATED"
   }
}
```

### Image Set Update Failed

#### 州-**UPDATE_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Update Failed",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
```

```
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATE_FAILED"
   }
}
```

## Image Set Deleting

### 州-**DELETING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleting",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "DELETING"
   }
}
```

## Image Set Deleted

### 州-**DELETED**

```
{
```

```
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleted",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "isPrimary": true,
      "imageSetVersion": "1",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "DELETED",
      "imageSetWorkflowStatus": "DELETED"
   }
}
```


### 图像集事件-元数据描述

| Name | Type | 说明 |
| --- | --- | --- |
| version | 字符串 | EventBridge 事件架构版本。 |
| id | 字符串 | 为每个事件生成的版本 4 UUID。 |
| detail-type | 字符串 | 正在发送的事件的类型。 |
| source | 字符串 | 标识生成事件的服务。 |
| account | 字符串 | 数据存储所有者的 12 位数 AWS 账户 ID。 |
| time | 字符串 | 事件发生的时间。 |
| region | 字符串 | 标识数据存储的 AWS 区域。 |

| Name | Type | 说明 |
| --- | --- | --- |
| resources | 数组（字符串） | 一个包含图像集的 ARN 的 JSON 数组。 |
| detail | object | 包含关于事件信息的 JSON 对象。 |
| detail.imagingVersion | 字符串 | 跟踪事件详细信息架构变更 HealthImaging的版本 ID。 |
| detail.isPrimary | 布尔值 | 表示导入的数据是否已成功组织到托管层次结构中，或者是否存在需要解决的元数据冲突。 |
| detail.imageSetVersion | 字符串 | 多次导入实例时，影像集版本将递增。最新版本将覆盖存储在主映像集中的任何旧版本。 |
| detail.datastoreId | 字符串 | 生成状态更改事件的数据存储 ID。 |
| detail.imagesetId | 字符串 | 与状态更改事件关联的图像集 ID。 |
| detail.imageSetState | 字符串 | 当前影像集的状态。 |
| detail.imageSetWorkflowStatus | 字符串 | 当前影像集工作流程状态。 |

# AWS 中的安全 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。要了解适用于 AWS 的合规计划 HealthImaging，请参阅按合规计划提供的范围内的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。以下主题向您展示如何进行配置 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
- 适用于 AWS 的 Identity and Access 管理 HealthImaging
- AWS 的合规性验证 HealthImaging
- AWS 中的基础设施安全 HealthImaging
- 使用创建 AWS HealthImaging 资源 AWS CloudFormation
- AWS HealthImaging 和接口 VPC 终端节点 (AWS PrivateLink)
- 的跨账户导入 AWS HealthImaging
- AWS 中的弹性 HealthImaging

# 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结构上的内容的控制。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解答AWS条款。 有关欧洲数据保护的信息，请参阅通用数据保护条例（GDPR）中心。

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。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）。
- 用于 SSL/TLS 与 AWS 资源通信。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。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 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 140-3 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 140-3 版》https://aws.amazon.com/compliance/fips/。

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 URL，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
- 网络流量隐私


## 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。这些方法包括：

- 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

默认情况下，使用服务拥有 HealthImaging AWS Key Management Service 的密钥对客户静态数据进行加密。或者，您可以配置 HealthImaging 为使用您创建、拥有和管理的对称客户管理 AWS KMS 密钥对静态数据进行加密。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加密 KMS 密钥。

## 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 拥有的 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）

如果您想完全控制 AWS KMS 生命周期和使用情况，则 HealthImaging 支持使用由您创建、拥有和管理的对称客户托管 KMS 密钥。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
- 安排密钥删除

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 管理控制台 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。

密钥政策控制对客户托管密钥的访问。每个客户托管式密钥必须只有一个密钥策略，其中包含确定谁可以使用密钥以及如何使用密钥的声明。创建客户托管式密钥时，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",
        "kms:GenerateDataKeyWithoutPlaintext"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:EncryptionContext:kms-arn": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f",
            "kms:EncryptionContext:aws:medical-imaging:datastoreId": "datastoreId"
        }
    }
}
```

## 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 账户必须拥有以下策略的权限，以及 AWS 的必要权限 HealthImaging。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:GenerateDataKey",
        "kms:RetireGrant",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f"
```

```
    }
  ]
}
```

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。

监控您的加密密钥 HealthImaging

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。

```
{
            "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
            "GrantId":
 "44e88bc45b769499ce5ec4abd5ecb27eeb3b178a4782452aae65fe885ee5ba20",
            "Name": "MedicalImagingGrantForQIDO_ebff634a-2d16-4046-9238-e3dc4ab54d29",
            "CreationDate": "2025-04-17T20:12:49+00:00",
            "GranteePrincipal": "AWS Internal",
            "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
```

```
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "ReEncryptFrom",
                "ReEncryptTo",
                "CreateGrant",
                "RetireGrant",
                "DescribeKey"
            ]
        },
        {
            "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
            "GrantId":
 "9e5fd5ba7812daf75be4a86efb2b1920d6c0c9c0b19781549556bf2ff98953a1",
            "Name": "2025-04-17T20:12:38",
            "CreationDate": "2025-04-17T20:12:38+00:00",
            "GranteePrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "IssuingAccount": "AWS Internal",
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "ReEncryptFrom",
                "ReEncryptTo",
                "CreateGrant",
                "RetireGrant",
                "DescribeKey"
            ]
        },
        {
            "KeyId": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c",
            "GrantId":
 "ab4a9b919f6ca8eb2bd08ee72475658ee76cfc639f721c9caaa3a148941bcd16",
            "Name": "9d060e5b5d4144a895e9b24901088ca5",
            "CreationDate": "2025-04-17T20:12:39+00:00",
            "GranteePrincipal": "AWS Internal",
            "RetiringPrincipal": "medical-imaging.us-east-1.amazonaws.com",
            "IssuingAccount": "medical-imaging.us-east-1.amazonaws.com",
```

```
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "ReEncryptFrom",
                "ReEncryptTo",
                "DescribeKey"
            ],
            "Constraints": {
                "EncryptionContextSubset": {
                    "kms-arn": "arn:aws:kms:us-
east-1:147997158357:key/8e1c34df-5fd2-49fa-8986-4618c9829a8c"
                }
            }
        }
```

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"
}
```

了解详情

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
- 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) 会加密与 A HealthImaging mazon S3 之间以及访问该存储桶的客户应用程序之间 HealthImaging 传输的对象，您应该使用 Amazon S `aws:SecureTransport condition`3 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 合作
- AWS 基于身份的策略示例 HealthImaging
- AWS AWS 的托管策略 HealthImaging
- 跨服务混淆了副手预防 HealthImaging
- 将服务相关角色用于 HealthImaging
- 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) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅对 AWS HealthImaging 身份和访问进行故障排除）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅AWS 如何 HealthImaging 与 IAM 合作）
- IAM 管理员：编写用于管理访问权限的策略（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 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center（例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问，AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

### AWS 账户 root 用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，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，要求人类用户使用与身份提供商的联合身份验证才能 AWS 服务 使用临时证书进行访问。

联合身份是指来自您的企业目录、Web 身份提供商的用户 Directory Service ，或者 AWS 服务 使用来自身份源的凭据进行访问的用户。联合身份代入可提供临时凭证的角色。

要集中管理访问权限，建议使用。 AWS IAM Identity Center有关更多信息，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

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

### IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

## 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。策略定义了与身份或资源关联时的权限。 AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

### 基于身份的策略

基于身份的策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

基于身份的策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管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。示例包括 IAM 角色信任策略和 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。您必须在基于资源的策略中指定主体。

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。

### 其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs)-在中指定组织或组织单位的最大权限 AWS Organizations。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs)-设置账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略 – 在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。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 合作

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 |
| ACLs | 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 |
| 服务关联角色 | 是 |

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

HealthImaging 支持指定 DICOM 研究和系列的条件语句 UIDs，从而允许访问控制范围限于一个或多个 DICOM 研究或系列。

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》中的使用客户管理型策略定义自定义 IAM 权限。

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。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 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。

要启用跨账户访问，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。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

## 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。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《服务授权参考》 HealthImaging中的 AWS 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：

```
medical-imaging
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "medical-imaging:action1",
```

```
      "medical-imaging: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。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于不支持资源级权限的操作，请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 HealthImaging 资源类型及其列表 ARNs，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。

`Condition` 元素根据定义的条件指定语句何时执行。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。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

## ACLs in HealthImaging

支持 ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 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 根据用户的工作职能定义权限，在 AWS 之外称为角色。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）：部分支持

> **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。

基于属性的访问权限控制（ABAC）是一种授权策略，该策略基于称为标签的属性来定义权限。您可以将标签附加到 IAM 实体和 AWS 资源，然后设计 ABAC 策略以允许在委托人的标签与资源上的标签匹配时进行操作。

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》中的使用 ABAC 授权定义权限。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）。

## 将临时证书与 HealthImaging

支持临时凭证：是

临时证书提供对 AWS 资源的短期访问权限，并且是在您使用联合身份或切换角色时自动创建的。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 用户指南》中的 IAM 中的临时安全凭证和使用 IAM 的。AWS 服务

## 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。

## 的服务角色 HealthImaging

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

支持服务关联角色：是

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关联角色的权限。

HealthImaging 使用服务相关角色向您的账户发布 CloudWatch 指标。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。

# AWS 基于身份的策略示例 HealthImaging

默认情况下，用户和角色没有创建或修改 HealthImaging 资源的权限。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。

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（控制台）。

有关 Awesome 定义的操作和资源类型（包括每种资源类型的格式）的详细信息，请参阅《服务授权参考》中的 A AWS wesome 的操作、资源和条件密钥。 ARNs

主题

- 策略最佳实践
- 使用控制 HealthImaging 台
- 允许用户查看他们自己的权限
- 根据研究实例 UID 和系列实例 UID 授予权限

## 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，请遵循以下指南和建议：

- 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 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》中的使用 IAM Access Analyzer 验证策略。
- 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》中的使用 MFA 保护 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 中的安全最佳实践。

## 使用控制 HealthImaging 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": "*"
        }
    ]
}
```

## 根据研究实例 UID 和系列实例 UID 授予权限

HealthImaging DICOMWeb APIs 支持根据研究实例 UID 和系列实例 UID 授予对影像集的访问权限。您可以通过添加带有`StudyInstanceUID`和`SeriesInstanceUID`条件上下文密钥的条件语句来定义限制访问的 IAM 策略。

HealthImaging DICOMWeb APIs `StudyInstanceUID`用作必需参数的支持基于`StudyInstanceUID`密钥限制访问的 IAM 策略。同样 HealthImaging DICOMWeb APIs ，`SeriesInstanceUID`作为必需参数的使用支持带有`SeriesInstanceUID`密钥的策略。

HealthImaging APIs 支持使用**`StudyInstanceUID`**和**`SeriesInstanceUID`**上下文密钥的 IAM 策略

| Name | 对**`StudyInstanceUID`** 条件的支持 | 对**`SeriesInstanceUID`** 条件的支持 |
| --- | --- | --- |
| `GetDICOMInstance` | 支持 | 是 |

| Name | 对**StudyInstanceUID** 条件的支持 | 对**SeriesInstanceUID** 条件的支持 |
| --- | --- | --- |
| GetDICOMInstanceFrames | 是 | 是 |
| GetDICOMInstanceMetadata | 是 | 是 |
| GetDICOMSeriesMetadata | 是 | 是 |
| GetDICOMBulkdata | 是 | 是 |
| SearchDICOMSeries | 是 | 否 |
| SearchDICOMInstances | 是 | 是 |
| StoreDICOMStudy | 是 | 否 |

> **Note**
>
> 当使用包含StudyInstanceUID或上下文密钥的策略调用时，不支持此上下文密钥的 HealthImaging API 将像未指定SeriesInstanceUID上下文密钥一样运行。

示例 1：根据 StudyInstance UID 授予访问权限

要仅授予对特定 DICOM 研究的访问权限，请为角色附加一个策略，该策略指定了StudyInstanceUID条件。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMSeries"
            ],
```

```
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-
id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:StudyInstanceUID": "your study instance UID"
                }
            }
        }
    ]
}
```

当通过担任此角色时sts assume-role，调用者将仅有权访问与角色策略中指定的条件相匹配的图像集，否则调用将被拒绝，并抛AccessDenied出错误。在这种情况下，调用者将被授予访问所有具有指定图像集的权限StudyInstanceUID。

您可以在策略中使用所有 IAM 字符串条件运算符，包括通配符匹配和多重匹配。

通配符匹配策略示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMSeries"
            ],
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-
id"
            ],
            "Condition": {
                "StringLike": {
                    "medical-imaging:StudyInstanceUID": "123.456.789*"
                }
            }
        }
    ]
}
```

多场比赛的策略示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMSeries"
            ],
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:StudyInstanceUID": [
                        "123.456.789",
                        "1.2.3.4.5.6"
                    ]
                }
            }
        }
    ]
}
```

示例 2：根据 SeriesInstance UID 授予访问权限

要仅授予对应于 DICOM 系列的特定影像集的访问权限，请将策略附加到指定条件的角色上`SeriesInstanceUID`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMInstances"
            ],
            "Resource": [
```

```
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-
id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:SeriesInstanceUID": [
                        "123.456.789",
                        "1.2.3.4.5.6"
                    ]
                }
            }
        }
    ]
}
```

示例 3：根据 StudyInstanceUIDs 和授予访问权限 SeriesInstance UIDs

要仅授予对特定 DICOM 研究和系列影像集的访问权限，请为角色附加一个策略，指定StudyInstanceUID和SeriesInstanceUID的条件。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "medical-imaging:SearchDICOMInstances"
            ],
            "Resource": [
                "arn:aws:medical-imaging:us-west-2:account-id:datastore/your-datastore-
id"
            ],
            "Condition": {
                "StringEquals": {
                    "medical-imaging:StudyInstanceUID": ["123.456.789"],
                    "medical-imaging:SeriesInstanceUID": ["1.2.3.4.5.6"]
                }
            }
        }
    ]
}
```

# AWS AWS 的托管策略 HealthImaging

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管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 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于使用案例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 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

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： AWSHealthImagingServiceRolePolicy
- AWS 托管策略： AWSHealthImagingFullAccess
- AWS 托管策略： AWSHealthImagingReadOnlyAccess
- HealthImaging AWS 托管策略的更新

## AWS 托管策略： AWSHealthImagingServiceRolePolicy

此策略附加到服务相关角色`AWSServiceRoleForHealthImaging`。它授 HealthImaging 予管理服务操作和发布服务指标的权限。

有关此策略的更多信息，包括 JSON 策略文档，请参阅 AWS 托管策略参考指南AWSHealthImagingServiceRolePolicy中的。

## AWS 托管策略： AWSHealthImagingFullAccess

您可以将 `AWSHealthImagingFullAccess` 策略附加到 IAM 身份。

此政策授予所有 HealthImaging 操作的管理权限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "medical-imaging:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "medical-imaging.amazonaws.com"
            }
        }
        }
    ]
}
```

## AWS 托管策略： AWSHealthImagingReadOnlyAccess

您可以将 `AWSHealthImagingReadOnlyAccess` 策略附加到 IAM 身份。

此策略向特定 AWS HealthImaging 操作授予只读权限。

JSON

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

```
            "medical-imaging:GetDICOMImportJob",
            "medical-imaging:GetDatastore",
            "medical-imaging:GetImageFrame",
            "medical-imaging:GetImageSet",
            "medical-imaging:GetImageSetMetadata",
            "medical-imaging:ListDICOMImportJobs",
            "medical-imaging:ListDatastores",
            "medical-imaging:ListImageSetVersions",
            "medical-imaging:ListTagsForResource",
            "medical-imaging:SearchImageSets"
        ],
        "Resource": "*"
    }]
}
```

## HealthImaging AWS 托管策略的更新

查看 HealthImaging 自该服务开始跟踪这些更改以来 AWS 托管策略更新的详细信息。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 | 描述 | 日期 |
| --- | --- | --- |
| AWSHealthImagingServiceRolePolicy | AWS 为服务相关角色 HealthImaging 添加了新的托管策略，该策略 HealthImaging 为管理服务操作和发布服务指标提供了权限。 | 2026年2月9日 |
| HealthImaging 已开始跟踪更改 | HealthImaging 开始跟踪其 AWS 托管策略的更改。 | 2023 年 7 月 19 日 |

# 跨服务混淆了副手预防 HealthImaging

混淆代理问题是一个安全性问题，即不具有某操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在 AWS 中，跨服务模拟可能会导致混淆副手问题。一个服务（呼叫服务）调用另一项服务

（所谓的服务）时，可能会发生跨服务模拟。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务不应有访问权限。为了防止这种情况，AWS 提供了一些工具，可帮助您保护所有服务委托人的数据，这些服务委托人已被授予访问您账户中资源的权限。

我们建议在您的 ImportJobDataAccessRole IAM 角色信任关系策略中使用aws:SourceArn和aws:SourceAccount全局条件上下文密钥来限制 AWS HealthImaging 为您的资源提供其他服务的权限。使用 aws:SourceArn 来仅将一个资源与跨服务访问相关联。使用 aws:SourceAccount 来让该账户中的任何资源与跨服务使用相关联。如果您同时使用两个全局条件上下文密钥，则在同一策略语句中使用 aws:SourceAccount 值和 aws:SourceArn 值中引用的账户时，必须使用相同的账户 ID。

的值aws:SourceArn必须是受影响数据存储的 ARN。如果您不知道数据存储的完整 ARN，或者要指定多个数据存储，请使用带有* 通配符的aws:SourceArn全局上下文条件键来表示 ARN 的未知部分。例如，您可以将 aws:SourceArn 设置为 arn:aws:medical-imaging:us-west-2:111122223333:datastore/*。

在以下信任策略示例中，我们使用aws:SourceArn和aws:SourceAccount条件键根据数据存储的 ARN 限制对服务主体的访问，以防止出现混淆的代理问题。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "medical-imaging.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
```

```
}
```

# 将服务相关角色用于 HealthImaging

AWS HealthImaging 使用服务预定义的 AWS Identity and Access Management (IAM) 服务相关角色，包括该服务代表您调用其他 AWS 服务所需的所有权限。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

## 的服务相关角色权限 HealthImaging

HealthImaging 使用名为AWSServiceRoleForHealthImaging的服务相关角色在您的 AWS 账户中执行操作。如果要将有关数据存储的指标发布 HealthImaging 到，则需要创建此服务相关角色。CloudWatch

名为的角色权限策略AWSHealthImagingServiceRolePolicy授 HealthImaging 予管理服务操作和发布服务指标的权限。

有关托管策略更新，请参阅HealthImaging 托管策略。

## 为创建服务相关角色 HealthImaging

使用 IAM 控制台创建服务相关角色

您可以使用 IAM 控制台创建服务相关角色，方法是选择 “AWS Service可信实体” 类型，然后HealthImaging在 “用例” 下拉菜单中选择。

使用 AWS CLI 创建服务相关角色

在 AWS CLI 中，运行 `aws iam create-service-linked-role –aws-service-name medical-imaging.amazonaws.com`

## 删除的服务相关角色 HealthImaging

您可以随时删除服务相关角色，但这样做会 HealthImaging 阻止您在 AWS 账户中执行操作，例如向CloudWatch发布数据存储指标。

使用 IAM 手动删除服务关联角色

您可以使用 IAM 控制台、AWS CLI 或 AWS API 来删除AWSServiceRoleForHealthImaging服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务关联角色。如果您删除了服务相关角色，则可以使用角色创建过程来创建新角色。

# 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，指明您无权执行某个操作，则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

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新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 `marymajor` 的 IAM 用户尝试使用控制台在 HealthImaging 中执行操作时，会发生以下示例错误。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，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

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 合作。
- 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 中的跨账户资源访问。

## AWS 的合规性验证 HealthImaging

HealthImaging 作为多项合规计划的一部分，第三方审计师评估 AWS 的安全与 AWS 合规性。对于 HealthImaging，这包括 HIPAA。

有关特定合规计划范围内的 AWS 服务列表，请参阅按合规计划划分的 AWS 范围内的服务 AWS 按合规计划。有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的下载报告 AWS Artifact。

您在使用 AWS HealthImaging 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- AWS 合作伙伴解决方案 —《安全与合规性自动参考部署指南》讨论了架构注意事项，并提供了在上部署以安全性和合规性为重点的基准环境的步骤。 AWS

- HIPAA 安全与合规架构白皮书 — 本白皮书描述了公司如何使用来 AWS 创建符合 HIPAA 标准的应用程序。
- GxP Systems on AWS — 本白皮书提供了有关如何处理 AWS 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 合规资源 — 此工作簿和指南集可能适用于您所在的行业和所在地。
- 根据@@ 规则评估资源- AWS Config 评估您的资源配置在多大程度上符合内部实践、行业指导方针和法规。
- AWS Security Hub CSPM— 此 AWS 服务可全面了解您的安全状态 AWS ，帮助您检查是否符合安全行业标准和最佳实践。

# AWS 中的基础设施安全 HealthImaging

作为一项托管服务，AWS HealthImaging 受《A mazon Web Services：安全流程概述》白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的 API 调用 HealthImaging 通过网络进行访问。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。

# 使用创建 AWS HealthImaging 资源 AWS CloudFormation

HealthImaging AWS 与 AWS CloudFormation一项服务集成，可帮助您对 AWS 资源进行建模和设置，从而减少创建和管理资源和基础设施所花费的时间。您可以创建一个描述所需的所有 AWS 资源的模板，并为您 CloudFormation 预置和配置这些资源。

使用时 CloudFormation，您可以重复使用模板来一致且重复地设置 HealthImaging 资源。只需描述一次您的资源，然后在多个 AWS 账户 区域中一遍又一遍地配置相同的资源。

## HealthImaging 和 CloudFormation 模板

要为和相关服务配置 HealthImaging 和配置资源，必须了解CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述了您要在 CloudFormation 堆栈中配置的资源。如果你不熟悉

JSON 或 YAML，可以使用 D CloudFormation esigner 来帮助你开始使用 CloudFormation 模板。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的什么是 CloudFormation Designer？。

AWS HealthImaging 支持使用创建数据存储 CloudFormation。有关更多信息，包括用于配置 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。

## 了解更多关于 CloudFormation

要了解更多信息 CloudFormation，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- 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 AWS PrivateLink， HealthImaging APIs 无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接即可使用该技术进行私密访问。您的 VPC 中的实例不需要公有 IP 地址即可与之通信 HealthImaging APIs。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 User Guide》中的 Creating an interface endpoint。

HealthImaging 使用以下服务名称创建 VPC 终端节点：

- com.amazonaws。 *region*. 医学影像
- com.amazonaws。 *region*。 runtime-medical-imaging
- com.amazonaws。 *region*。 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

您可以为 VPC 端点附加控制对 HealthImaging 的访问的端点策略。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 中的弹性 HealthImaging

AWS 全球基础设施是围绕 AWS 区域 可用区构建的。 AWS 区域 提供多个物理分隔和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。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失效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、容错能力和可扩展性。

如果需要跨更大的地理距离复制数据或应用程序，请使用 AWS 本地区域。 AWS 本地区域是旨在补充现有 AWS 区域的单一数据中心。像所有区域一样 AWS 区域， AWS 本地区域与其他区域完全隔离 AWS 区域。

有关 AWS 区域 和可用区的更多信息，请参阅AWS 全球基础设施。

# AWS HealthImaging 参考资料

> **Note**
>
> AWS HealthImaging API 参考中描述了所有原生 HealthImaging API 操作和数据类型。

主题


- DICOM 对 AWS 的支持 HealthImaging
- AWS HealthImaging 参考资料


## 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


- 支持的 SOP 课程
- 元数据标准化
- 支持的传输语法
- 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 支持使用不同的传输语法导入 DICOM P10 文件。有些文件在导入期间会保留其原始传输语法编码，而大多数无损图像帧是在导入期间进行转码的。转码行为取决于您的数据存储配置。默认情况下，数据存储使用 HTJ2 K 作为存储格式，但可以在创建时将其配置为使用 JPEG 2000 Lossless。以下示例显示了如何在返回`StoredTransferSyntaxUID`的元数据中 HealthImaging 记录每个实例的`GetImageSetMetadata`。

```
"Instances": {
   "999.999.2.19941105.134500.2.101": {
       "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.90",
       "ImageFrames": [{ ...
```

> **Note**
>
> 查看下表时，请记住以下几点：
>
> - 标有星号 (*) 的传输语法 UID 条目表示文件在导入期间以其原始编码格式存储。对于这些文件，StoredTransferSyntaxUID位于实例元数据中的与原始传输语法相匹配。
> - 不带星号的传输语法 UID 条目表示文件在导入过程中被转码为 HTJ2 K 无损并存储在中。HealthImaging实例元数据的StoredTransferSyntaxUID元素将设置为带有 RPCL 选项图像压缩的高吞吐量 JPEG 2000（仅限无损）（1.2.840.10008.1.2.4.202）的存储格式。
> - 如果StoredTransferSyntaxUID密钥不存在或设置为null，则可以假设它已编码为配置的数据存储存储格式。

HealthImaging 支持的传输语法

| 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*（二进制分割保留原始编码，而非二进制分段则转码为 K Lossless RPCL）HTJ2 | 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） |

| 传输语法 UID | 传输语法名称 |
| --- | --- |
| 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 |
| 1.2.840.10008.1.2.4.92 | JPEG 2000 第 2 部分多分量图像压缩（仅限无损） |
| 1.2.840.10008.1.2.4.93 | JPEG 2000 第 2 部分多分量图像压缩 |
| 1.2.840.10008.1.2.4.112* | JPEG XL |
| 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 |
| 1.2.840.10008.1.2.4.100*，1.2.840.10008.1.2.4.100.1* | MPEG2 主要配置文件主级别 |
| 1.2.840.10008.1.2.4.101*，1.2.840.10008.1.2.4.101.1* | MPEG2 高层主要简介 |
| 1.2.840.10008.1.2.4.102*，1.2.840.10008.1.2.4.102.1* | MPEG-4 AVC/H.264 High Profile/4.1 级 |
| 1.2.840.10008.1.2.4.103*，1.2.840.10008.1.2.4.103.1* | MPEG-4 AVC/H.264 BD 兼容 High Profile /4.1 级 |

| 传输语法 UID | 传输语法名称 |
| --- | --- |
| 1.2.840.10008.1.2.4.104*，1.2.840.10008.1.2.4.104.1* | MPEG-4 AVC/H.264 High Profile /适用于 2D 视频的 4.2 级 |
| 1.2.840.10008.1.2.4.105*，1.2.840.10008.1.2.4.105.1* | MPEG-4 AVC/H.264 High Profile/ITU-T H.264 视频的 4.2 级 |
| 1.2.840.10008.1.2.4.106*，1.2.840.10008.1.2.4.106.1* | MPEG-4 AVC/H.264 Stereo High Profile /ITU-T H.264 视频的 4.2 级 |
| 1.2.840.10008.1.2.4.107* | HEVC/H.265 主要配置文件/5.1 级视频 |
| 1.2.840.10008.1.2.4.108* | HEVC/H.265 主要 10 配置文件/等级 5.1 |

*导入期间保留原始传输语法编码

## DICOM 元素限制

将您的医学影像数据导入 AWS 时 HealthImaging，将对以下 DICOM 元素施加最大长度限制。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。

导入期间的 DICOM 元素约束

| DICOM 关键字 | DICOM 标签 | 最大长度 |
| --- | --- | --- |
| PatientName | (0010,0010) | 256 |
| patienTid | (0010,0020) | 256 |
| PatientBirthDate | (0010,0030) | 18 |
| PatientSex | (0010,0040) | 16 |
| StudyInstanceUID | (0020,000D) | 256 |
| StudyID | (0020,0010) | 256 |
| StudyDescription | (0008,1030) | 256 |
| NumberOfStudyRelatedSeries | (0020,1206) | 1000000 |

| DICOM 关键字 | DICOM 标签 | 最大长度 |
| --- | --- | --- |
| NumberOfStudyRelatedInstances | (0020,1208) | 1000000 |
| AccessionNumber | (0008,0050) | 256 |
| StudyDate | (0008,0020) | 18 |
| StudyTime | (0008,0030) | 28 |
| SOPInstanceUID | (0008,0018) | 256 |
| SeriesInstanceUID | (0020,000E) | 256 |

## 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。

- 除非更新约束同时`updatableAttributes`适用于和，否则无法更新或移除Patient/Study/Series/Instance关卡属性的私有属性 `removableAttributes`
- 无法更新以下 AWS HealthImaging 生成的属性：`SchemaVersionDatastoreID`、`ImageSetID`、`PixelData`、`Checksum`、`Width`、、`Height`、`M`
- 除非设置了`force`标志，否则无法更新以下 DICOM 属性：`Tag.PixelData`、、`Tag.StudyInstanceUID`、`Tag.SeriesInstanceUID`、`Tag.SOPInstance` `Tag.StudyID`
- 除非设置了`force`标志，否则无法更新 VR 类型 SQ 的属性（嵌套属性）
- 除非设置了`force`标志，否则无法更新多值属性
- 除非设置了`force`标志，否则无法使用与 VR 属性类型不兼容的值来更新属性
- 除非设置了标`force`志，否则无法更新根据 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 参考资料

本部分包含与 AWS 相关的支持数据 HealthImaging。

主题


- AWS HealthImaging 终端节点和配额
- AWS HealthImaging 限制限制
- AWS HealthImaging 像素数据验证
- HealthImaging 警告码
- AWS 的图像帧解码库 HealthImaging
- AWS HealthImaging 示例项目
- 将此服务与 AWS SDK


## 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
- 服务配额


### 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 | |
| | | medical-imaging-fips.us-east-1.amazonaws.com | HTTPS | |

| 区域名称 | 区域 | 端点 | 协议 |
| --- | --- | --- | --- |
| 美国西部（俄勒冈州） | us-west-2 | medical-imaging.us-west-2.amazonaws.com<br>medical-imaging-fips.us-west-2.amazonaws.com | HTTPS<br>HTTPS |
| 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 |
| 欧洲地区（伦敦） | eu-west-2 | medical-imaging.eu-west-2.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

可通过端点访问以下数据存储、导入和标记操作：

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- StartDICOMImportJob

- GetDICOMImportJob
- ListDICOMImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

可通过端点访问以下映像集操作：

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging 提供 DICOMWeb 检索WADO-RS 服务的表示。有关更多信息，请参阅 正在从中检索 DICOM 数据 HealthImaging。

以下 DICOMWeb 服务可通过端点访问：

```
https://dicom-medical-imaging.region.amazonaws.com
```

- `GetDICOMInstance`
- `GetDICOMInstanceMetadata`
- `GetDICOMInstanceFrames`

## 服务配额

服务配额定义为 AWS 账户中资源、操作和项目的最大价值。

> **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。

| Name | 默认值 | 可调整 | 说明 |
| --- | --- | --- | --- |
| 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 |

| Name | 默认值 | 可调整 | 说明 |
| --- | --- | --- | --- |
| 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<br><br>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 |
| 最大数据存储 | 每个受支持的区域：100 个 | 是 | 当前 AWS 区域中活跃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 区域中每个 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：5000 个 | 是 | 当前 AWS 区域中 DICOM 导入任务的最大文件数 |
| 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 导入任务中的最大嵌套文件夹数 |
| 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负载大小限制（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） |

| Name | 默认值 | 可调整 | 说明 |
| --- | --- | --- | --- |
| 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：10 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 每次导入、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。要创建案例，请登录您的 AWS 账户并选择创建案例。

下表列出了本机 HealthImaging 操作和服务表示的 DICOMweb 限制限制。

AWS HealthImaging 限制限制

| Action | 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 |

| Action | 限制率 | 节流突增 |
| --- | --- | --- |
| 开始 DICOMImport Job | 1 tps | 2 tps |
| Get DICOMImport Job | 25 tps | 50 tps |
| 列出DICOMImport职位 | 10 tps | 20 tps |
| SearchImageSets | 25 tps | 50 tps |
| GetImageSet | 25 tps | 50 tps |
| GetImageSetMetadata | 50 tps | 100 tps |
| GetImageFrame | 1000 tps | 2000 tps |
| ListImageSetVersions | 25 tps | 50 tps |
| UpdateImageSetMetadata | 0.25 tps | 1 tps |
| CopyImageSet | 0.25 tps | 1 tps |
| DeleteImageSet | 0.25 tps | 1 tps |
| TagResource | 10 tps | 20 tps |
| ListTagsForResource | 10 tps | 20 tps |
| UntagResource | 10 tps | 20 tps |
| 获取 DICOMInstance * | 50 tps | 100 tps |
| 获取DICOMInstance元数据* | 50 tps | 100 tps |
| 获取DICOMInstance帧* | 50 tps | 100 tps |
| 获取DICOMSeries元数据 | 50 tps | 100 tps |

*服务代表 DICOMweb

# AWS HealthImaging 像素数据验证

在导入过程中，通过检查每张图像的无损编码和解码状态， HealthImaging 提供内置的像素数据验证。此功能可确保使用 HTJ2K 解码库解码的图像始终与导入的原始 D ICOM P10 图像相匹配。HealthImaging

- 当导入任务在导入 DICOM P10 图像之前捕获它们的原始像素质量状态时，图像加载过程就开始了。使用该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)。 CRC32 IFRC 校验和值显示在`job-output-manifest.json`元数据文档中。有关更多信息，请参阅 了解导入任务。
- 将图像导入 HealthImaging 数据存储并转换为图像集后，将立即解码 HTJ2 K编码的图像帧并计算出新的 IFRCs 图像帧。 HealthImaging 然后将原始图像 IFRCs 的全分辨率与导入的新 IFRCs 图像帧进行比较，以验证准确性。
- 导入任务输出日志 (`job-output-manifest.json`) 中会捕获相应的每张图像描述性错误情况，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）。有关更多信息，请参阅 了解导入任务。
2. 影像集由元数据和图像框（像素数据）组成。图像集元数据包含有关关联图像帧的信息。使用`GetImageSetMetadata`操作获取影像集的元数据。有关更多信息，请参阅 获取影像集元数据。
3. `PixelDataChecksumFromBaseToFullResolution`包含全分辨率图像的 IFRC（校验和）。对于以原始传输语法 1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50 和 1.2.840.10008.1.2.1（仅限二进制分割）存储的图像，校验和是在原始图像上计算的。对于使用 RPCL 存储在 HTJ2 K Lossless 中的图像，校验和是在解码后的全分辨率图像上计算的。有关更多信息，请参阅 支持的传输语法。

   以下是 IFRC 的元数据输出示例，该输出是在导入任务过程中生成并记录到`job-output-manifest.json`的。

```
"ImageFrames": [{
"ID": "678906789067890123451234512345 12",
"PixelDataChecksumFromBaseToFullResolution": [
{
    "Width": 512,
    "Height": 512,
    "Checksum": 2510355201
```

```
}
]
```

对于以原始传输语法 1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50 和 1.2.840.10008.1.2.1（仅限二进制分割）存储的图像，和将不可用。`MinPixelValue` `MaxPixelValue``FrameSizeInBytes`表示原始框架的大小。

```
"PixelDataChecksumFromBaseToFullResolution": [
   {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": null,
"MaxPixelValue": null,
"FrameSizeInBytes": 429
```

对于使用 RPCL 存储在 HTJ2 K Lossless 中的图像，`FrameSizeInBytes`表示解码后的图像帧的大小。

```
"PixelDataChecksumFromBaseToFullResolution": [
   {"Width": 512, "Height": 512, "Checksum": 1379921327 }
],
"MinPixelValue": 11,
"MaxPixelValue": 11,
"FrameSizeInBytes": 1652
```

4. 对于包含视频的实例， HealthImaging 执行轻量级编解码器验证，以验证 DICOM 元数据中指定的传输语法与视频编解码器是否匹配。

   HealthImaging 使用算法计算原始视频对象上的 IFRC 校验和值。 CRC32IFRC 的校验和值记录到并保留在元数据中。`job-output-manifest.json` HealthImaging 与以原始传输语法（如上所述）存储的图像一样，`MinPixelValue`和`MaxPixelValue`将不可用。 FrameSizeInBytes 表示原始框架的大小。

5. HealthImaging 验证像素数据，访问像素数据验证程序， GitHub 然后按照`README.md`文件中的说明独立验证所图像帧解码库使用的各种无损图像处理。 HealthImaging加载完整图像后，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 IFRC 值进行比较以验证像素数据。

# HealthImaging 警告码

HealthImaging 尝试导入您的所有医学影像数据。如果在导入过程中遇到数据不一致或无法识别的数据元素，则 HealthImaging 将在文件中添加以下警告之一。`warning.ndjson`还可以通过对`WarningReason`元素执行的`SearchDICOMInstances`操作来搜索与导入的实例相关的警告。如下所述，带有警告的导入实例可能会通过 HealthImaging APIs减少支持。

HealthImaging 导入警告码

| 警告原因（十六进制） | 警告原因（十进制） | 警告类型（枚举） | 警告详情 | 产生的行为 |
|---|---|---|---|---|
| DICOM 标准警告原因 | | | | |
| 0xB000 | 45056 | 强制使用数据元素 | 在实例存储期间，摄取修改了一个或多个数据元素。参见第 6.6.1.3 节。 | 不适用 |
| 0xb006 | 45062 | 元素_已丢弃 | 在实例存储期间，摄取丢弃了一些数据元素。参见第 6.6.1.3 节。 | 不适用 |
| 0xb007 | 45063 | SOP_CLASS_DATA_不匹配 | 该`StoreDICOM` 操作发现，在实例存储期间，数据集与 SOP 类别的限制不匹配。 | 不适用 |
| AWS HealthImaging 警告原因 | | | | |
| 0xb100 | 45312 | 转码异常 | 当 HealthImaging 无法 PixelData 将实例转码为 HTJ2 K（默认存储格式），或者由于其他原因（即验证失败、像素数据属性错误等）而 PixelData 无法对实例进行转码时，就会出现此警告。在这种情 | 如果需要，像素数据仍然可以作为单个 blob 检索，传递通配符 “*” 作为 Accept 标头将`transfer-syntax` 以存储的格式返回。 |

| 警告原因（十六进制） | 警告原因（十进制） | 警告类型（枚举） | 警告详情 | 产生的行为 |
| --- | --- | --- | --- | --- |
| | | | 况下，像素数据将存储为 blob。 | |
| 0xb110 | 45328 | 帧提取失败 | 当 PixelData 根据给定的 DICOM 元数据解析单个帧时出现问题时，就会出现此警告。 | 像素数据格式不正确，无法检索，`GetDICOMInstance` 用于检索整个实例 |
| 0xb111 | 45329 | 帧数不匹配 | 当 “NumberOfFrames” DICOM 元素与输入 DICOM 文件中图像 “片段” 的实际数量不匹配时，就会出现此警告。 | 像素数据格式不正确，无法检索，`GetDICOMInstance` 用于检索整个实例 |
| 0xb112 | 45330 | 无效偏移表 | 当输入 DICOM 文件片段中的偏移表与实际帧长度不匹配并且根据严重程度可能导致帧格式错误时，就会出现此警告。 | 像素数据格式不正确，无法检索，`GetDICOMInstance` 用于检索整个实例 |
| 0xb120 | 45344 | 不支持的传输语法 | 当 HealthImaging 遇到无法识别或不支持的传输语法时，就会出现此警告。发生这种情况时，HealthImaging 会将像素数据存储为 blob。 | 如果需要，像素数据仍然可以作为单个 blob 检索，传递通配符 “*” 作为 Accept 标头将`transfer-syntax` 以存储的格式返回。 |
| 0xb201 | 45570 | UID_FORMAT 无效 | 当一个或多个 UID 元素违反 DICOM 值表示法时（例如），就会出现此警告 `1.2.3..4` | 不适用 |

| 警告原因（十六进制） | 警告原因（十进制） | 警告类型（枚举） | 警告详情 | 产生的行为 |
|---|---|---|---|---|
| 0xb202 | 45571 | 无效的 DICOM_VALUE_LENGTH | 当 DICOM 元素的长度超过 DICOM 值表示所支持的长度时，就会出现此警告，这可能会引入无效的 search/retrieve 操作行为。 | 某些字段可能无法解析，因此无法搜索（即StudyDate 或）StudyTime |
| 0xbfff | 47513 | OTHER | 当有未捕获的警告 HealthImaging 未捕获为特定警告代码时，就会出现此警告。 | 像素数据格式不正确，无法检索，GetDICOMInstance 用于检索整个实例 |

## AWS 的图像帧解码库 HealthImaging

在导入过程中，某些传输语法会保留其原始编码，而另一些则根据您的数据存储配置，默认转码为高吞吐量 JPEG 2000 (HTJ2K) 无损或 JPEG 2000 Lossless（配置后）。 HTJ2K 提供始终如一的快速图像显示和对 HTJ2 K 高级功能的通用访问。由于图像帧在导入过程中以 HTJ2 K 或 JPEG 2000 Lossless 编码，因此在图像查看器中查看之前必须对其进行解码。有关确定传输语法的信息，请参阅支持的传输语法。有关确定传输语法的信息，请参见支持的传输语法。

> **Note**
>
> HTJ2K 在 JPEG2000 标准（ISO/IEC 15444-15:2019）的第 15 部分中定义。 HTJ2K 保留了 JPEG2 000 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

- 图像帧解码库
- 图像查看器

### 图像帧解码库

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 nv JPEG2 000 — 商用、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 示例项目

AWS 在上 HealthImaging 提供了以下示例项目 GitHub。

OHIF Viewer HealthImaging 通过 OIDC 集成到 AWS

该AWS Cloud Development Kit (AWS CDK)项目在亚马逊上部署 OHIF 查看器。 CloudFront查看器作为数据源集成到亚马逊网络服务数据存储中，A ma zon Cognito 作为 DICOMWeb 身份提供者进行身份验证，通过 OIDC 进行身份验证。

DICOM 从本地接入到 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 系列排队等候导入 AWS。HealthImaging它由在边缘运行且由AWS IoT Greengrass管理的组件和在云中运行的 DICOM 摄取管道组成。 AWS

方块等级标记 (TLM) 代理

一个 HealthImaging 通过使用高吞吐量 JPEG 2000 (K) 的功能——切片级标记 (TLM) 从 AWS 检索图像帧的AWS Cloud Development Kit (AWS CDK)项目。HTJ2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 终端节点的 Amazon CloudFront 分配，该终端节点缓存（使用 GET）并从边缘传送图像帧。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 CloudFront 是 Amazon 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 查看器用户界面

一个用于部署带有后端身份验证的前端用户界面的AWS Amplify项目，您可以使用该用户界面 HealthImaging 使用渐进式解码查看存储在 AWS 中的图像集元数据属性和图像框架（像素数据）。您可以选择使用其他方法集成上面的图块级别标记 (TLM) 代理 and/or Amazon Del CloudFront ivery 项目，以加载图像框架。

AWS HealthImaging DICOMweb 代理

一个基于 Python 的项目，用于在 HealthImaging数据存储上启用 DICOMweb WADO-RS 和 QIDO-RS 端点，以支持基于 Web 的医学成像查看器和其他兼容的应用程序。 DICOMweb

> **Note**
>
> 此项目未使用中 DICOMweb APIs 描述 HealthImaging的表示形式在 AW DICOMweb S 中使用 HealthImaging。

要查看其他示例项目，请参阅上的 AWS HealthImaging 示例 GitHub。

## 将此服务与 AWS SDK

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 |
| --- | --- |
| 适用于 C++ 的 AWS SDK | 适用于 C++ 的 AWS SDK 代码示例 |

| SDK 文档 | 代码示例 |
| --- | --- |
| AWS CLI | AWS CLI 代码示例 |
| 适用于 Go 的 AWS SDK | 适用于 Go 的 AWS SDK 代码示例 |
| 适用于 Java 的 AWS SDK | 适用于 Java 的 AWS SDK 代码示例 |
| 适用于 JavaScript 的 AWS SDK | 适用于 JavaScript 的 AWS SDK 代码示例 |
| 适用于 Kotlin 的 AWS SDK | 适用于 Kotlin 的 AWS SDK 代码示例 |
| 适用于 .NET 的 AWS SDK | 适用于 .NET 的 AWS SDK 代码示例 |
| 适用于 PHP 的 AWS SDK | 适用于 PHP 的 AWS SDK 代码示例 |
| AWS Tools for PowerShell | AWS Tools for PowerShell 代码示例 |
| AWS 适用于 Python 的 SDK 版本 | AWS 适用于 Python 的 SDK v4 代码示例 |
| 适用于 Python (Boto3) 的 AWS SDK | 适用于 Python (Boto3) 的 AWS SDK 代码示例 |
| 适用于 Ruby 的 AWS SDK | 适用于 Ruby 的 AWS SDK 代码示例 |
| 适用于 Rust 的 AWS SDK | 适用于 Rust 的 AWS SDK 代码示例 |
| 适用于 SAP ABAP 的 AWS SDK | 适用于 SAP ABAP 的 AWS SDK 代码示例 |
| 适用于 Swift 的 AWS SDK | 适用于 Swift 的 AWS SDK 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。

# 成本优化

HealthImaging 旨在通过在访问模式发生变化时自动将数据移动到最经济实惠的访问层来优化存储成本。随着使用模式随时间的推移而变化，智能分层可为 DICOM 数据提供自动生命周期管理，而不会产生任何运营开销。 HealthImaging 有两个存储层：

- 频繁访问层存储，用于存放新导入或经常访问的 DICOM 数据。
- 将最近未@@ 访问过的 DICOM 数据存档即时访问层存储。降低了长期存档的存储成本，同时保持了毫秒级的访问延迟。

您需要为数据存储中所有影像集的聚合存储大小付费。两个存储层均按每月每 GB 计费，并且不收取每个映像集的费用。在存储层之间移动数据也不会产生检索费用。

> **Note**
>
> 图像集的最小大小为 5 MB 计费。 HealthImaging 接受小于 5 MB 的图像集，但这些较小的对象按照 5 MB 的费率收费。

## 智能分层的工作原理

创建新影像集时，您的数据将存储在频繁访问层中。可以通过导入新数据（通过导入任务或 DICOMweb STOW-RS）或通过调用来创建图像集。 CopyImageSet HealthImaging 自动将连续 30 天未被访问的影像集移至存档即时访问层，并且影像集将保留在存档即时访问层中，直到再次被访问为止。

以下是构成对 DICOM 数据的访问权限的操作，这些操作会自动将影像集从存档即时访问层移回频繁访问层：

- 调用GetImageSetMetadataGetImageFrame、或任何 DICOMweb WADO-RS 操作。
- 调用CopyImageSet或。UpdateImageSetMetadata对于复制操作，只有复制的映像集才会分层到频繁访问级别。如果使用目标复制，则目标影像集会向上分层。
- 通过 AWS HealthImaging 管理控制台查看和下载影像集数据。

上述操作既可以将映像集提升到频繁访问层，又可以防止频繁访问层中的映像集在接下来的 30 天内向下分层到存档即时访问层。可以通过 AWS 管理控制台或编程接口（例如或）访问影像集 AWS

SDKs。 AWS CLI 其他操作不构成访问，因此不会自动将对象从 “存档即时访问” 层移回 “频繁访问” 层。以下是此类操作的示例，而不是最终列表：

- 对数据存储的操作（CreateDatastore和 GetDatastore）
- 列出操作（列出DICOMImport作业ListImageSetVersions、和 ListTagsForResource）
- 搜索操作（SearchImageSets以及 DICOMweb QIDO-RS 查询）
- 调用GetImageSetTagResource、或 UntagResource
- 删除操作

导入到 HealthImaging 的数据的最短存储时间为 30 天。您可以随时删除数据，但在导入后 30 天之前删除的每张图像都将按最短时限的剩余时间收费。

## 估算结构化数据存储

HealthImaging 将一些 DICOM 标头信息存储在索引存储中。您需要为这种结构化存储消耗付费，与图像集存储层无关，而且这些费用是累加的。您可以通过将数据存储中的 DICOM 资源数量乘以每个资源的索引记录大小来估算结构化存储消耗。以下值为近似值。

| DICOM 资源 | 索引大小（字节） |
| --- | --- |
| 研究 | 1024 |
| 系列 | 830 |
| 实例 | 680 |

# 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。有关版本的更多信息，请参阅链接的主题。

| 变更 | 说明 | 日期 |
| --- | --- | --- |
| 对 QIDO-RS 搜索 API 的关系查询支持 | 借 HealthImaging助，您现在可以将 QIDO-RS 关系查询与三个新 API 一起使用：`SearchDICOMStudyInstances` `SearchDICOMAllSeries` 、和`SearchDICOMAllInstances` 。使用关系查询，您无需在层次结构中指定所有父级 UID 即可搜索系列和实例。有关更多信息，请参阅搜索 DICOM 数据。 | 2026 年 6 月 30 日 |
| QIDO-RS 搜索 API 的扩展分页功能 | 借助 HealthImaging，您现在可以在 QIDO-RS 搜索 API 中对超过 10,000 个结果进行分页。最大抵消量已从9,000增加到100万个。现在，您可以访问大型结果集，例如包含超过 10,000 条匹配记录的系列或实例查询。有关更多信息，请参阅搜索 DICOM 数据。 | 2026 年 6 月 30 日 |
| `includefield` QIDO-RS 搜索 API 的查询参数 | HealthImaging 现在支持所有三个 QIDO-RS 搜索 API（`SearchDICOMStudies` `SearchDICOMSeries` 、和`SearchDICOMInstances` ）中的`includefield` 查询参 | 2026年5月22日 |

| | | |
|---|---|---|
| | 数。此功能允许您请求默认响应集之外的其他 DICOM 属性，包括标准标签、使用虚线路径表示法的嵌套序列 (SQ) 属性和私有数据元素。您也可以指定检索`includefield=all` 给定资源级别的所有可用属性。有关更多信息，请参阅搜索 DICOM 数据。 | |
| 导入 DICOM 的 JSON 元数据 `StartDICOMImportJob` | HealthImaging 将`importConfiguration` 参数添加到`StartDICOMImportJob` 操作中。现在，您可以通过提供将 DICOM 文件映射到相应的 JSON 元数据文件的`DicomMetadataMapping` ，在导入过程中使用 JSON 元数据增强现有的 DICOM 文件。有关更多信息，请参阅 StartDICOMImportJob。 | 2026 年 5 月 20 日 |
| 原子研究级别的更新 `UpdateImageSetMetadata` | HealthImaging 为`UpdateImageSetMetadata` 动作添加`--include-study-image-sets` 标志。设置后，对指定影像集所做的患者和 Study-level 属性更改将自动应用于数据存储中共享相同研究实例 UID 的所有其他主影像集。有关更多信息，请参阅更新影像集元数据。 | 2026年3月31日 |

| | | |
|---|---|---|
| Study-level 精细的访问控制 | HealthImaging 现在支持直接在 IAM 策略中使用 DICOM 研究实例 UID 和系列实例 UID 为 DicomWeb API 授予权限。有关更多信息，请参阅根据研究实例 UID 和系列实例 UID 授予权限。 | 2026 年 3 月 19 日 |
| 以下各项的亚马逊 CloudWatch 指标 HealthImaging | HealthImaging 向`AWS/HealthImaging` 命名空间 CloudWatch 中发布其他指标，包括账户和数据存储级别的资源使用量指标。有关更多信息，请参阅将 Amazon CloudWatch 与配合使用 HealthImaging。 | 2026 年 2 月 12 日 |
| Service-linked 的角色 HealthImaging | HealthImaging 现在支持服务相关角色，允许服务代表您向发布数据存储指标。 CloudWatch 可以使用 IAM 控制台或 AWS CLI 创建该`AWSServiceRoleForHealthImaging` 角色。有关更多信息，请参阅为 HealthImaging 使用服务相关角色。 | 2026年2月9日 |
| AWSHealthImagingServiceRolePolicy 管理的策略 | HealthImaging `AWSHealthImagingServiceRolePolicy` 为服务相关角色添加了新的托管策略，该策略提供管理服务操作和向其发布服务指标的权限。 CloudWatch有关更多信息，请参阅 AWSHealthImagingServiceRolePolicy。 | 2026年2月9日 |

| | | |
|---|---|---|
| 对 AWS HealthImaging 数据存储的 JPEG XL 支持 | HealthImaging 支持以 JPEG XL 传输语法 (1.2.840.10008.1.2.4.112) 导入和存储 DICOM 有损文件。有关更多信息，请参阅支持的传输语法。 | 2026 年 2 月 2 日 |
| 增强了 DICOM 导入功能，支持对不合格数据发出警告 | HealthImaging 现在通过接受不符合要求的文件来导入以前被拒绝的 DICOM 数据，同时通过事件提供详细警告。EventBridge 现在，导入作业会生成一个 WARNING `warning.ndjson` 文件夹，其中包含针对不合格数据的特定警告原因代码的文件。此更新增加了可搜索 WarningReason (0008,1196) DICOM 元素支持，并引入了在转码不可行时以存储格式检索实例的`transfer-syntax=*` 参数。有关更多信息，请参阅了解导入任务和HealthImaging 警告代码。 | 2025 年 12 月 8 日 |
| JPEG 2000 对 AWS 数据存储的无损支持 HealthImaging | AWS HealthImaging 现在支持医疗图像的原生 JPEG 2000 无损编码，使您无需转码即可创建保留和检索 JPEG 2000 格式的无损图像帧的数据存储。当您在创建数据存储时指定`–lossless-storage-format JPEG_2000_LOSSLESS` 时，这可以降低需要 JPEG 2000 无损格式的应用程序的检索延迟。 | 2025 年 11 月 25 日 |

| | | |
|---|---|---|
| QIDO-RS 搜索增强功能设计 | HealthImaging 现在支持对关键 DICOM 属性（患者姓名、转诊医生姓名、患者 ID、研究描述、模式和注册号）进行通配符搜索，并支持 Patient/Referring 医生姓名的模糊搜索功能，以适应不完整或拼写错误的术语。此次更新还扩展了 QIDO-RS API查询功能，将患者出生日期和研究描述搜索包括在内，从而增强了查找相关研究和系列的能力。有关更多信息，请参阅中 HealthImaging 搜索 DICOM 数据。 | 2025 年 9 月 15 日 |
| DicomWeb API 的 OpenID Connect (OIDC) 授权 | HealthImaging 现在支持所有 DicomWeb API 的 OpenID Connect (OIDC) 持有者令牌授权。通过在标题中接受 JWT，这增加了与普通查看器和工具包（例如 OHIF、SLIM、MONAI）的基于标准的 OAuth 2.0 互操作性。 IdP-issued `Authorization` 有关更多信息，请参阅 DicomWeb API 的 OIDC 身份验证。 | 2025 年 9 月 3 日 |
| 支持 dicomWeb 数据导入和 dicomWeb Bulkdata | HealthImaging HealthImaging 支持通过 dicomWeb 协议存储 DICOM P10 文件。 STOW-RS 有关更多详细信息，请参阅导入数据。此外，还 HealthImaging 支持 DICOM Bulkdata，以确保一致的低延迟元数据检索。有关更多信息，请参阅获取 DICOM 批量数据。 | 2025 年 6 月 30 日 |

| | | |
|---|---|---|
| 自动数据组织、DicomWeb QIDO-RS 搜索和 dicomWeb 增强功能 WADO-RS | HealthImaging 根据 DICOM 标准的患者、研究和系列级别层次结构，在导入时自动整理 DICOM P10 数据。有关更多信息，请参阅了解导入任务。 HealthImaging 根据 DicomWeb QIDO-RS 标准，支持丰富的搜索。有关更多信息，请参阅中 HealthImaging 搜索 DICOM 数据。 HealthImaging 支持通过单个 API 操作检索系列中所有 DICOM 实例的元数据，如从中获取 DICOM 系列元数据中所述。 HealthImaging | 2025 年 5 月 22 日 |
| 图像集创建使用更少的 DICOM 元素 | HealthImaging 减少将传入的 DICOM P10 对象分组为图像集时使用的元素数量。有关更多信息，请参阅什么是影像集？。 | 2025 年 1 月 27 日 |
| 有损支持 StartDICOMImportJob | HealthImaging 支持以原始格式导入和存储 DICOM 有损文件（1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50）和二进制分段文件。有关更多信息，请参阅支持的传输语法。 | 2024 年 11 月 1 日 |

| | | |
|---|---|---|
| 对 DicomWeb 检索 API 的有损支持 | HealthImaging 支持检索存储在 1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.50 和 1.2.840.10008.1.2.1（仅限二进制分割）中的图像和实例，其原始格式或显式 VR Little Endian（1.2.840.10008.1.2.1）。有关更多信息，请参阅支持的传输语法和检索 DICOM 数据。 | 2024 年 11 月 1 日 |
| 更快地导入数字病理学 | HealthImaging 支持 DICOM 数字病理学 (WSI) 的导入作业速度提高多达 6 倍。 | 2024 年 11 月 1 日 |
| 二进制分割支持 | HealthImaging 支持摄取和检索 DICOM 二进制分段文件。有关更多信息，请参阅支持的传输语法。 | 2024 年 11 月 1 日 |
| 恢复到之前的影像集版本 ID | HealthImaging 提供了用于恢复到先前影像集版本 ID 的`revertToVersionId` 参数。有关更多信息，请参阅 AWS HealthImaging API 参考*`revertToVersionId`* 中的。 | 2024 年 7 月 24 日 |

| | | |
|---|---|---|
| 用于修改影像集的强制功能 | HealthImaging 为Overrides 数据类型提供可选的forced请求参数。即使患者、研究或系列级别的元数据不匹配，设置此参数也会强制CopyImageSet 执行UpdateImageSetMetadata 和操作。有关更多信息，请参阅 AWS HealthImaging API 参考中的替代。<br>• UpdateImageSetMetadata 强制功能-HealthImaging 引入了用于更新以下属性的可选force请求参数：<br>• Tag.StudyInstanceUID 、Tag.SeriesInstanceUID 、Tag.SOPInstanceUID 和Tag.StudyID<br>• 添加、移除或更新实例级私有 DICOM 数据元素<br>有关更多信息，请参阅 AWS HealthImaging API 参考 UpdateImageSetMetadata中的。<br>• CopyImageSet 强制功能-HealthImaging 引入了用于复制影像集的可选force请求参数。即使患者、研究或系列级别的元数据在sourceImageSet 和destinati | 2024 年 7 月 24 日 |

| | | |
|---|---|---|
| | onImageSet 之间不匹配，设置此参数也会强制执行CopyImageSet 操作。在这些情况下，不一致的元数据在中保持不变destinationImageSet 。有关更多信息，请参阅 AWS HealthImaging API 参考CopyImageSet中的。 | |
| 复制 SOP 实例的子集 | HealthImaging 增强了CopyImageSet 操作，因此您可以从 a 中选择一个或多个 SOP 实例sourceImageSet 以复制到destinationImageSet 。有关更多信息，请参阅复制影像集。 | 2024 年 7 月 24 日 |
| GetDICOMInstanceMetadata 用于返回 DICOM 实例元数据 | HealthImaging 提供了返回 DICOM 第 10 部分元数据（.json文件）的 GetDICOMInstanceMetadata API。有关更多信息，请参阅获取实例元数据。 | 2024 年 7 月 11 日 |
| GetDICOMInstanceFrames 用于返回 DICOM 实例帧（像素数据） | HealthImaging 提供了返回 DICOM 第 10 部分帧（multipart 请求）的 GetDICOMInstanceFrames API。有关更多信息，请参阅获取实例帧。 | 2024 年 7 月 11 日 |

| | | |
|---|---|---|
| 增强了对非标准 DICOM 数据导入的支持 | HealthImaging 为包含与 DICOM 标准偏差的数据导入提供支持。有关更多信息，请参阅 DICOM 元素约束。<br>• 以下 DICOM 数据元素的最大长度可以为 256 个字符：<br>  • `Patient's Name (0010,0010)`<br>  • `Patient ID (0010,0020)`<br>  • `Accession Number (0008,0050)`<br>• 、、、、和 `Study Instance UID Series Instance UID Treatment Session UIDManufacturer's Device Class UID`，允许使用以下语法变`Device UID`体`Acquisition UID`：<br>  • 任何 UID 的第一个元素都可以为零<br>  • UID 可以以一个或多个前导零开头<br>  • UID 的长度最多可为 256 个字符 | 2024 年 6 月 28 日 |
| 事件通知 | HealthImaging 与 Amazon 集成 EventBridge 以支持事件驱动的应用程序。有关更多信息，请参阅使用 EventBridge。 | 2024 年 6 月 5 日 |

| | | |
|---|---|---|
| GetDICOMInstance 用于返回 DICOM 实例数据 | HealthImaging 提供返回 DICOM 第 10 部分实例数据（.dcm文件）的GetDICOMInstance 服务。有关更多信息，请参阅获取实例。 | 2024 年 5 月 15 日 |
| Cross-account 进口 | 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Cross-account导入。 | 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。<br>• 对搜索UpdatedAt 和 的额外支持 SeriesInstanceUID<br>• 在开始时间和结束时间之间进行搜索<br>• 按Ascending 或对搜索结果进行排序 Descending<br>• 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 文件的最大文件大小为 4 GB。有关更多信息，请参阅服务配额。 | 2024 年 3 月 6 日 |

| | | |
|---|---|---|
| JPEG Lossless 和 HTJ2K 的传输语法 | HealthImaging 为任务导入提供以下传输语法的支持。有关更多信息，请参阅支持的传输语法。<br>• 1.2.840.10008.1.2.4.57 — JPEG Lossless（进程 14）Non-Hierarchical<br>• 1.2.840.10008.1.2.4.201 — JPEG 2000 图像压缩（仅限无损）High-Throughput<br>• 1.2.840.10008.1.2.4.202 — 带有 RPCL 选项图像压缩的 High-Throughput JPEG 2000（仅限无损）<br>• 1.2.840.10008.1.2.4.203 — JPEG 2000 图像压缩 High-Throughput | 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，请参阅服务配额。 | 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，请参阅服务配额。 | 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，请参阅服务终端节点。 | 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，请参阅使用创建 HealthImaging 资源 CloudFormation。 | 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。有关更多信息，请参阅服务终端节点。 | 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。