



開發人員指南

# AWS HealthImaging



# AWS HealthImaging: 開發人員指南

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

# Table of Contents

什麼是 AWS HealthImaging ? .....	1
重要通知 .....	2
功能 .....	2
相關服務 .....	3
存取 .....	4
HIPAA .....	4
定價 .....	5
開始使用 .....	6
概念 .....	6
資料存放區 .....	6
影像集 .....	6
中繼資料 .....	7
影像影格 .....	7
設定 .....	7
註冊 AWS 帳戶 .....	7
建立具有管理存取權的使用者 .....	8
建立 S3 儲存貯體 .....	9
建立資料存放區 .....	10
建立 IAM 使用者 .....	10
建立 IAM 角色 .....	11
安裝 AWS CLI .....	13
教學課程 .....	14
管理資料存放區 .....	15
建立資料存放區 .....	15
取得資料存放區屬性 .....	22
列出資料存放區 .....	28
刪除資料存放區 .....	35
了解儲存層 .....	41
匯入影像資料 .....	44
了解匯入任務 .....	44
啟動匯入任務 .....	47
取得匯入任務屬性 .....	54
列出匯入任務 .....	60
存取映像集 .....	66

了解映像集 .....	66
搜尋映像集 .....	73
取得映像集屬性 .....	97
取得映像集中繼資料 .....	103
取得影像集像素資料 .....	112
修改映像集 .....	120
列出映像集版本 .....	120
更新影像集中繼資料 .....	126
	144
複製映像集 .....	146
刪除映像集 .....	160
<b>標記 資源 .....</b>	<b>166</b>
<b>標記資源 .....</b>	<b>166</b>
<b>列出資源的標籤 .....</b>	<b>171</b>
<b>取消標記資源 .....</b>	<b>175</b>
<b>程式碼範例 .....</b>	<b>181</b>
<b>基本概念 .....</b>	<b>187</b>
Hello HealthImaging .....	188
動作 .....	194
<b>案例 .....</b>	<b>328</b>
<b>影像集和影像影格入門 .....</b>	<b>329</b>
<b>標記資料存放區 .....</b>	<b>383</b>
<b>標記影像集 .....</b>	<b>393</b>
<b>DICOMweb .....</b>	<b>404</b>
<b>擷取資料 .....</b>	<b>404</b>
<b>取得執行個體 .....</b>	<b>406</b>
<b>取得執行個體中繼資料 .....</b>	<b>407</b>
<b>取得執行個體影格 .....</b>	<b>409</b>
<b>取得序列中繼資料 .....</b>	<b>411</b>
<b>搜尋資料 .....</b>	<b>412</b>
<b>HealthImaging 的 DICOMweb 搜尋 API APIs .....</b>	<b>413</b>
<b>HealthImaging 支援的 DICOMweb 查詢類型 .....</b>	<b>413</b>
<b>搜尋研究 .....</b>	<b>415</b>
<b>搜尋系列 .....</b>	<b>416</b>
<b>搜尋執行個體 .....</b>	<b>417</b>
<b>監控 .....</b>	<b>419</b>

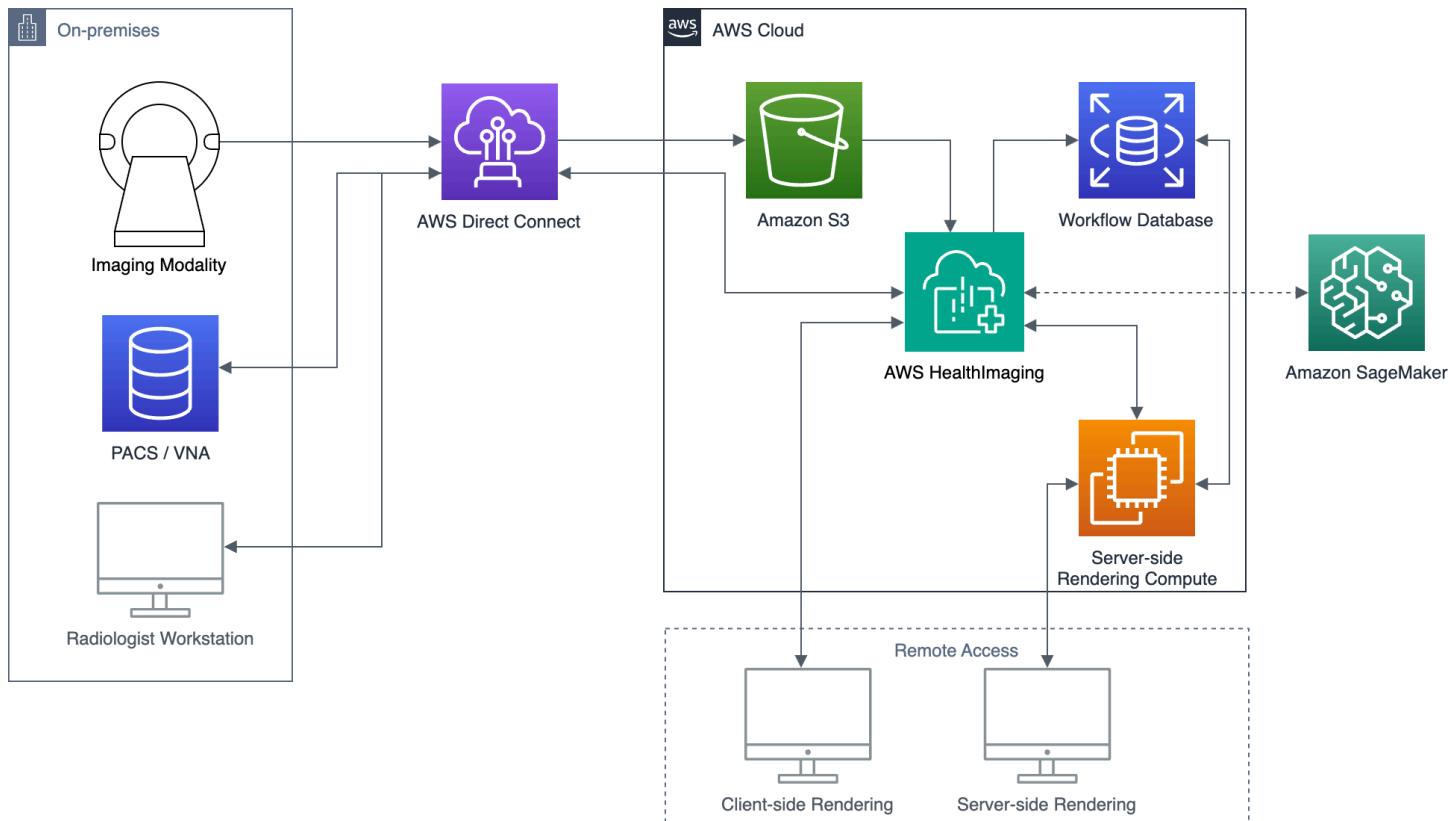
CloudTrail (API 呼叫 ) .....	419
建立追蹤 .....	420
了解日誌項目 .....	421
CloudWatch ( 指標 ) .....	422
檢視 HealthImaging 指標 .....	423
建立警報 .....	423
EventBridge ( 事件 ) .....	424
HealthImaging 事件傳送至 EventBridge .....	424
HealthImaging 事件結構和範例 .....	425
安全 .....	441
資料保護 .....	441
資料加密 .....	442
網路流量隱私權 .....	452
身分和存取權管理 .....	452
目標對象 .....	453
使用身分驗證 .....	453
使用政策管理存取權 .....	456
AWS HealthImaging 如何與 IAM 搭配使用 .....	458
身分型政策範例 .....	464
AWS 受管政策 .....	466
預防跨服務混淆代理人 .....	468
故障診斷 .....	469
法規遵循驗證 .....	471
基礎架構安全 .....	472
基礎設施即程式碼 .....	472
HealthImaging 和 AWS CloudFormation 範本 .....	472
進一步了解 AWS CloudFormation .....	472
VPC 端點 .....	473
VPC 端點的考量事項 .....	473
建立一個 VPC 端點 .....	473
建立 VPC 端點政策 .....	474
跨帳戶匯入 .....	475
恢復能力 .....	476
參考資料 .....	478
DICOM .....	478
支援的 SOP 類別 .....	478

中繼資料標準化 .....	479
支援的傳輸語法 .....	483
DICOM 元素限制條件 .....	486
DICOM 中繼資料限制條件 .....	487
<b>HealthImaging .....</b>	<b>488</b>
端點和配額 .....	488
調節限制 .....	493
像素資料驗證 .....	495
HTJ2K 解碼程式庫 .....	497
範例專案 .....	498
使用 AWS SDKs .....	499
<b>推出 .....</b>	<b>501</b>
	dix

# 什麼是 AWS HealthImaging？

AWS HealthImaging 是一項符合 HIPAA 資格的服務，可讓醫療保健供應商、生命科學組織及其軟體合作夥伴以 PB 級規模在雲端中存放、分析和共用醫療影像。HealthImaging 使用案例包括：

- 企業影像 – 直接從 AWS 雲端存放和串流醫療影像資料，同時保持低延遲效能和高可用性。
- 長期映像封存 – 節省長期映像封存的成本，同時維持次秒的映像擷取存取。
- AI/ML 開發 – 透過其他工具和服務的支援，對影像存檔執行人工智慧和機器學習 (AI/ML) 推論。
- 多模態分析 – 將臨床影像資料與 AWS HealthLake (運作狀態資料) 和 AWS HealthOmics (數值資料) 結合，以提供精準醫學的洞見。



AWS HealthImaging 可讓您存取影像資料（例如 X-Ray、CT、MRI、超音波），以便建置在雲端中的醫療影像應用程式先前只能在內部部署實現效能。使用 HealthImaging，您可以從 中每個醫療影像的單一授權副本大規模執行醫療影像應用程式，以降低基礎設施成本 AWS 雲端。

## 主題

- [重要通知](#)

- [AWS HealthImaging 的功能](#)
- [相關 AWS 服務](#)
- [存取 AWS HealthImaging](#)
- [HIPAA 資格和資料安全](#)
- [定價](#)

## 重要通知

AWS HealthImaging 無法取代專業醫療建議、診斷或治療，也無法修復、治療、緩解、預防或診斷任何疾病或健康狀況。您負責在 AWS HealthImaging 的任何使用過程中進行人工審核，包括與旨在通知臨床決策的任何第三方產品相關。AWS HealthImaging 只有在經過訓練的醫療專業人員採用健全的醫療判斷後，才能用於病患照護或臨床案例。

## AWS HealthImaging 的功能

AWS HealthImaging 提供下列功能。

### 開發人員易用的 DICOM 中繼資料

AWS HealthImaging 以開發人員易用的格式傳回 DICOM 中繼資料，簡化應用程式開發。匯入影像資料後，可以使用人性化關鍵字存取個別中繼資料屬性，而不是不熟悉的群組/元素十六進位數字。病患、研究和序列層級 DICOM 元素已標準化，無需應用程式開發人員處理 SOP 執行個體之間的不一致。此外，中繼資料屬性值可在原生執行時間類型中直接存取。

### SIMD 加速影像解碼

AWS HealthImaging 會傳回編碼為高輸送量 JPEG 2000 (HTJ2K) 影像的影像影格（像素資料），這是一種進階影像壓縮轉碼器。HTJ2K 利用現代處理器上的單一指令多個資料 (SIMD) 來提供新層級的效能。HTJ2K 是比 JPEG2000 快許多的順序，而且至少比所有其他 DICOM 傳輸語法快兩倍。WASM-SIMD 可用來將此極端速度帶入零足跡 Web 檢視器。如需詳細資訊，請參閱支援的傳輸語法。

### 像素資料驗證

AWS HealthImaging 透過在匯入期間檢查每個映像的無失真編碼和解碼狀態，提供內建像素資料驗證。如需詳細資訊，請參閱像素資料驗證。

## 領先業界的效能

AWS HealthImaging 有效率的中繼資料編碼、無失真壓縮和漸進式解析度資料存取，為映像載入效能設定了新標準。高效率中繼資料編碼可讓影像檢視器和 AI 演算法了解 DICOM 研究的內容，而不必載入影像資料。由於進階影像壓縮，影像載入速度更快，影像品質也不會受到影響。漸進式解析度可讓縮圖、關注區域和低解析度行動裝置的影像載入速度更快。

## 可擴展的 DICOM 匯入

AWS HealthImaging 匯入利用現代雲端原生技術平行匯入多個 DICOM 研究。歷史封存可以快速匯入，而不會影響新資料的臨床工作負載。如需支援的 SOP 執行個體和傳輸語法的相關資訊，請參閱 [DICOM](#)。

## 服務受管 DICOM 資料階層

AWS HealthImaging 會自動組織依病患、檢查和序列層級 DICOM 資料元素匯入的 DICOM P10 資料。此服務會將此 DICOM 資料整理成對應至 DICOM 系列的影像集，簡化匯入後工作流程。匯入新資料時，會維護檢查和序列層級組織。

## DICOMweb API 相容性

AWS HealthImaging 提供符合 DICOMweb 的 APIs，可簡化整合，並啟用與現有應用程式的互通性。此服務也提供雲端原生 APIs，可啟用 DICOMweb 標準不支援的動作，例如中繼資料更新操作。

## 相關 AWS 服務

AWS HealthImaging 與其他 AWS 服務緊密整合。了解下列服務有助於充分利用 HealthImaging。

- [AWS Identity and Access Management](#) – 使用 IAM 安全地管理身分和存取 HealthImaging 資源。
- [Amazon Simple Storage Service](#) – 使用 Amazon S3 做為暫存區域，將 DICOM 資料匯入 HealthImaging。
- [Amazon CloudWatch](#) – 使用 CloudWatch 觀察和監控 HealthImaging 資源。
- [AWS CloudTrail](#) – 使用 CloudTrail 追蹤 HealthImaging 使用者活動和 API 用量。
- [AWS CloudFormation](#) – 使用 AWS CloudFormation 實作基礎設施做為程式碼 (IaC) 範本，以在 HealthImaging 中建立資源。
- [AWS PrivateLink](#) – 使用 Amazon VPC 在 HealthImaging 和 [Amazon Virtual Private Cloud](#) 之間建立連線，而不會將資料公開至網際網路。

- [Amazon EventBridge](#) – 使用 EventBridge 透過建立將 HealthImaging 事件路由至目標的規則來建立可擴展的事件驅動型應用程式。

## 存取 AWS HealthImaging

您可以使用 AWS Management Console AWS Command Line Interface 和 AWS SDKs 存取 AWS HealthImaging。本指南提供 AWS Management Console 和 AWS CLI SDKs 程式碼範例 AWS 的程序說明。

### AWS Management Console

AWS Management Console 提供用於管理 HealthImaging 及其相關聯資源的 Web 型使用者介面。如果您已註冊 AWS 帳戶，您可以登入 [HealthImaging 主控台](#)。

### AWS Command Line Interface (AWS CLI)

為廣泛的 AWS 產品 AWS CLI 提供命令，並支援 Windows、Mac 和 Linux。如需詳細資訊，請參閱 [「AWS Command Line Interface 使用者指南」](#)。

### AWS SDKs

AWS SDKs 為軟體開發人員提供程式庫、程式碼範例和其他資源。這些程式庫提供基本函數，可自動化任務，例如以密碼編譯方式簽署您的請求、重試請求，以及處理錯誤回應。如需詳細資訊，請參閱 [要建置的工具 AWS](#)。

### HTTP 請求

您可以使用 HTTP 請求呼叫 HealthImaging 動作，但您必須根據使用的動作類型指定不同的端點。如需詳細資訊，請參閱 [HTTP 請求支援的 API 動作](#)。

## HIPAA 資格和資料安全

此為 HIPAA 合格服務。如需 1996 年 AWS 美國健康保險流通與責任法案 (HIPAA) 以及使用 AWS 服務來處理、存放和傳輸受保護醫療資訊 (PHI) 的詳細資訊，請參閱 [HIPAA 概觀](#)。

包含 PHI 和個人身分識別資訊 (PII) 的 HealthImaging 連線必須加密。根據預設，所有 HealthImaging 連線都會透過 TLS 使用 HTTPS。HealthImaging 會存放加密的客戶內容，並根據 [AWS 共同責任模型](#) 運作。

如需合規的相關資訊，請參閱 [AWS HealthImaging 的合規驗證](#)。

## 定價

HealthImaging 可協助您使用智慧型分層來自動化臨床資料的生命週期管理。如需詳細資訊，請參閱[了解儲存層](#)。

如需一般定價資訊，請參閱[AWS HealthImaging 定價](#)。若要預估成本，請使用[AWS HealthImaging 定價計算器](#)。

# AWS HealthImaging 入門

若要開始使用 AWS HealthImaging，請設定 AWS 帳戶並建立 AWS Identity and Access Management 使用者。若要使用 [AWS CLI](#) 或 [AWS SDKs](#)，您必須安裝和設定它們。

了解 HealthImaging 概念和設定後，您可以使用包含程式碼範例的簡短教學課程，協助您開始使用。

## 主題

- [AWS HealthImaging 概念](#)
- [設定 AWS HealthImaging](#)
- [AWS HealthImaging 教學課程](#)

## AWS HealthImaging 概念

下列術語和概念是您了解和使用 AWS HealthImaging 的核心。

### 概念

- [資料存放區](#)
- [影像集](#)
- [中繼資料](#)
- [影像影格](#)

## 資料存放區

資料存放區是醫療影像資料的儲存庫，位於單一 AWS 區域。AWS 帳戶可以有零或多個資料存放區。資料存放區有自己的 AWS KMS 加密金鑰，因此一個資料存放區中的資料可以與其他資料存放區中的資料進行實體和邏輯隔離。資料存放區支援使用 IAM 角色、許可和屬性型存取控制的存取控制。

如需詳細資訊，請參閱[管理資料存放區及了解儲存層](#)。

## 影像集

影像集是一種 AWS 概念，可定義摘要分組機制，以最佳化相關的醫療影像資料。當您將 DICOM P10 影像資料匯入 AWS HealthImaging 資料存放區時，它會轉換為包含[中繼資料](#)和[影像影格](#)的影像集（像素資料）。HealthImaging 會根據研究、系列和執行個體的 DICOM 階層嘗試組織匯入的資料。成功新增至 HealthImaging 受管階層的 DICOM 執行個體會標示為主要[影像集](#)。匯入 DICOM P10 資料將：建

立新的主要映像集；如果執行個體已存在於主要集合中，則將執行個體合併為現有的主要映像集；或者，如果中繼資料元素發生衝突，則建立新的非主要映像集。

如需詳細資訊，請參閱[匯入影像資料及了解映像集](#)。

## 中繼資料

中繼資料是影像集中存在的非像素屬性。對於 DICOM，這包括患者人口統計特性、程序詳細資訊和其他擷取特定參數。AWS HealthImaging 會將影像集分隔為中繼資料和影像影格（像素資料），讓應用程式可以快速存取影像集。這有助於不需要像素資料的影像檢視器、分析和 AI/ML 使用案例。DICOM 資料會在病患、檢查和序列層級標準化，消除不一致。這可簡化資料的使用、提高安全性並改善存取效能。

如需詳細資訊，請參閱[取得映像集中繼資料及中繼資料標準化](#)。

## 影像影格

影像影格是存在於影像集中以組成 2D 醫療影像的像素資料。有些檔案會在匯入期間保留其原始傳輸語法編碼，而其他檔案預設會轉碼為高輸送量 JPEG 2000 (HTJ2K) 無失真。如果影像影格以 HTJ2K 編碼，則必須先解碼，才能在影像檢視器中檢視。如需詳細資訊，請參閱[支援的傳輸語法](#)、[取得影像集像素資料及 HTJ2K 解碼程式庫](#)。

## 設定 AWS HealthImaging

您必須先設定 AWS 環境，才能使用 AWS HealthImaging。下列主題是下一節教學課程的先決條件。

### 主題

- [註冊 AWS 帳戶](#)
- [建立具有管理存取權的使用者](#)
- [建立 S3 儲存貯體](#)
- [建立資料存放區](#)
- [使用 HealthImaging 完整存取許可建立 IAM 使用者](#)
- [建立用於匯入的 IAM 角色](#)
- [安裝 AWS CLI \( 選用 \)](#)

## 註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

## 註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

註冊程序的一部分包括接聽電話或文字訊息，並在電話鍵盤上輸入驗證碼。

當您註冊時 AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 <https://aws.amazon.com/> 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

## 建立具有管理存取權的使用者

註冊後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center 和建立管理使用者，以免將根使用者用於日常任務。

### 保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者[AWS Management Console](#)身分登入。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者（主控台）啟用虛擬 MFA 裝置。

### 建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱 AWS IAM Identity Center《使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

## 以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM Identity Center 使用者登入的說明，請參閱 AWS 登入《使用者指南》中的[登入 AWS 存取入口網站](#)。

## 指派存取權給其他使用者

- 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[建立許可集](#)。

- 將使用者指派至群組，然後對該群組指派單一登入存取權。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[新增群組](#)。

## 建立 S3 儲存貯體

若要將 DICOM P10 資料匯入 AWS HealthImaging，建議使用兩個 Amazon S3 儲存貯體。Amazon S3 輸入儲存貯體存放要匯入的 DICOM P10 資料，且 HealthImaging 會從此儲存貯體讀取。Amazon S3 輸出儲存貯體會儲存匯入任務的處理結果，而 HealthImaging 會寫入此儲存貯體。如需視覺效果，請參閱的圖表[了解匯入任務](#)。

### Note

由於 AWS Identity and Access Management (IAM) 政策，您的 Amazon S3 儲存貯體名稱必須是唯一的。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[儲存貯體命名規則](#)。

為了本指南的目的，我們在 IAM 角色中指定下列 Amazon S3 輸入和輸出儲存貯體以進行匯入。[???](#)

- 輸入儲存貯體：`arn:aws:s3:::amzn-s3-demo-source-bucket`
- 輸出儲存貯體：`arn:aws:s3:::amzn-s3-demo-logging-bucket`

如需詳細資訊，請參閱《Amazon S3 使用者指南》中的[建立儲存貯體](#)。

## 建立資料存放區

當您匯入醫療影像資料時，AWS HealthImaging [資料存放區](#)會保留轉換後 DICOM P10 檔案的結果，稱為[影像集](#)。如需視覺效果，請參閱 [的圖表了解匯入任務](#)。

### Tip

datastoreID 會在您建立資料存放區時產生。在本節稍後完成[trust relationship](#)匯入datastoreID時，您必須使用。

若要建立資料存放區，請參閱 [建立資料存放區](#)。

## 使用 HealthImaging 完整存取許可建立 IAM 使用者

### 最佳實務

我們建議您針對匯入、資料存取和資料管理等不同需求建立個別的 IAM 使用者。這符合 AWS Well-Architected Framework 中的[授予最低權限存取](#)。

為達下一節[教學](#)的目的，您將使用單一 IAM 使用者。

## 建立 IAM 使用者

1. 請遵循《[IAM 使用者指南](#)》中在 AWS 帳戶中建立 IAM 使用者的指示。請考慮命名使用者 ahiadmin（或類似者）以進行釐清。
2. 將 AWSHealthImagingFullAccess 受管政策指派給 IAM 使用者。如需詳細資訊，請參閱[AWS 受管政策：AWSHealthImagingFullAccess](#)。

### Note

IAM 許可可以縮小範圍。如需詳細資訊，請參閱[AWS AWS HealthImaging 的受管政策](#)。

## 建立用於匯入的 IAM 角色

### Note

下列指示是指 AWS Identity and Access Management (IAM) 角色，可授予 Amazon S3 儲存貯體的讀取和寫入存取權，以匯入您的 DICOM 資料。雖然下一節的教學課程需要角色，但我們建議您使用將 IAM 許可新增至使用者、群組和角色AWS HealthImaging 的受管政策，因為它們比自行撰寫政策更容易使用。

IAM 角色是您可以在帳戶中建立的另一種 IAM 身分，具有特定的許可。若要啟動匯入任務，呼叫 StartDICOMImportJob 動作的 IAM 角色必須連接到使用者政策，該政策授予讀取 DICOM P10 資料和儲存匯入任務處理結果的 Amazon S3 儲存貯體存取權。它也必須指派信任關係（政策），讓 AWS HealthImaging 能夠擔任該角色。

### 建立用於匯入目的的 IAM 角色

1. 使用 [IAM 主控台](#) 建立名為的角色 ImportJobDataAccessRole。您可以在下一節的教學課程中使用此角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[建立 IAM 角色](#)。

### Tip

基於本指南的目的，中的程式碼範例會啟動匯入任務參考 IAM ImportJobDataAccessRole 角色。

2. 將 IAM 許可政策連接至 IAM 角色。此許可政策會授予 Amazon S3 輸入和輸出儲存貯體的存取權。將下列許可政策連接至 IAM 角色 ImportJobDataAccessRole。

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

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

3. 將下列信任關係（政策）連接至 IAM ImportJobDataAccessRole 角色。信任政策需要您完成 區段時datastoreId產生的 [建立資料存放區](#)。本主題後面的[教學](#)假設您使用一個 AWS HealthImaging 資料存放區，但使用資料存放區特定的 Amazon S3 儲存貯體、IAM 角色和信任政策。

 Note

此信任政策中的 Condition 區塊透過確保只能存取您的特定 AWS HealthImaging 資料存放區，協助防止混淆代理人問題。如需此安全措施的詳細資訊，請參閱 [HealthImaging 中的跨服務混淆代理人預防](#)。

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

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

若要進一步了解如何透過 AWS HealthImaging 建立和使用 IAM 政策，請參閱 [AWS HealthImaging 的 Identity and Access Management](#)。

若要進一步了解 IAM 角色，請參閱《IAM 使用者指南》中的 [IAM 角色](#)。若要進一步了解 IAM 政策和許可，請參閱《IAM 使用者指南》中的 [IAM 政策和許可](#)。

## 安裝 AWS CLI（選用）

如果您使用，則需要下列程序 AWS Command Line Interface。如果您使用的是 AWS Management Console AWS SDKs，則可以略過下列程序。

### 設定 AWS CLI

1. 下載和設定 AWS CLI。如需說明，請參閱《AWS Command Line Interface 使用者指南》中的下列主題。
  - [安裝或更新最新版本的 AWS CLI](#)
  - [開始使用 AWS CLI](#)
2. 在 AWS CLI config 檔案中，新增管理員的具名設定檔。您在執行 AWS CLI 命令時使用此設定檔。在最低權限的安全原則下，我們建議您建立具有所執行任務特定權限的個別 IAM 角色。如需具名設定檔的詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的 [組態和登入資料檔案設定](#)。

```
[default]
aws_access_key_id = default access key ID
```

```
aws_secret_access_key = default secret access key  
region = region
```

3. 使用以下help命令驗證設定。

```
aws medical-imaging help
```

如果 AWS CLI 設定正確，您會看到 AWS HealthImaging 的簡短描述和可用命令的清單。

## AWS HealthImaging 教學課程

### 目標

本教學課程的目標是將 DICOM P10 二進位檔 (.dcm 檔案) 匯入 AWS HealthImaging [資料存放區](#)，並將其轉換為包含[中繼資料](#)和[影像影格](#)（像素資料）的[影像集](#)。匯入 DICOM 資料後，您可以使用 HealthImaging 雲端原生動作，根據您的存取[偏好設定來存取](#)影像集、中繼資料和影像影格。

### 先決條件

所有列於 的程序[設定](#)都需要完成本教學課程。

### 教學步驟

1. [開始匯入任務](#)
2. [取得匯入任務屬性](#)
3. [搜尋映像集](#)
4. [取得映像集屬性](#)
5. [取得影像集中繼資料](#)
6. [取得影像集像素資料](#)
7. [刪除資料存放區](#)

# 使用 AWS HealthImaging 管理資料存放區

使用 AWS HealthImaging，您可以建立和管理醫療影像資源的資料存放區。下列主題說明如何使用 HealthImaging 雲端原生動作來使用 和 SDKs 建立、描述 AWS Management Console AWS CLI、列出和 AWS 刪除資料存放區。

## Note

本章的最後一個主題是了解儲存層。將醫療影像資料匯入 HealthImaging 資料存放區後，它會根據時間和用量，在兩個儲存層之間自動移動。儲存層有不同的定價層級，因此了解層移動程序和 HealthImaging 資源對於計費目的的認可非常重要。

## 主題

- [建立資料存放區](#)
- [取得資料存放區屬性](#)
- [列出資料存放區](#)
- [刪除資料存放區](#)
- [了解儲存層](#)

## 建立資料存放區

使用 CreateDatastore 動作來建立 AWS HealthImaging 資料存放區以匯入 DICOM P10 檔案。下列功能表提供 的程序 AWS Management Console，以及 AWS CLI 和 AWS SDKs 程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考[CreateDatastore](#)中的。

## 重要

請勿將資料存放區命名為受保護的健康資訊 (PHI)、個人身分識別資訊 (PII) 或其他機密或敏感資訊。

## 建立資料存放區

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

## AWS 主控台

1. 開啟 HealthImaging 主控台[建立資料存放區頁面](#)。
2. 在詳細資訊下，針對資料存放區名稱，輸入資料存放區的名稱。
3. 在資料加密下，選擇用於加密資源的 AWS KMS 金鑰。如需詳細資訊，請參閱[AWS HealthImaging 中的資料保護](#)。
4. 在標籤 - 選用下，您可以在建立標籤時將標籤新增至資料存放區。如需詳細資訊，請參閱[標記資源](#)。
5. 選擇建立資料存放區。

## AWS CLI 和 SDKs

### Bash

#### AWS CLI 使用 Bash 指令碼

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

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

```
function imaging_create_datastore() {
    local datastore_name response
    local option OPTARG # Required to use getopt command in a function.

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

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

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

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

    local error_code=${?}

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

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

echo "$response"

return 0
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 建立資料存放區

下列create-datastore程式碼範例會建立名為 的資料存放區my-datastore。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [建立資料存放區](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [CreateDatastore](#)。

## Java

### SDK for Java 2.x

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     datastoreStatus: 'CREATING'
    // }
    return response;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

```

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

# 取得資料存放區屬性

使用 `GetDatastore` 動作來擷取 AWS HealthImaging [資料存放](#) 區屬性。下列功能表提供 AWS Management Console 和 AWS CLI AWS SDKs 程式碼範例的程序。如需詳細資訊，請參閱 AWS HealthImaging API 參考[GetDatastore](#) 中的。

## 取得資料存放區屬性

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。
2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟。在詳細資訊區段下，所有資料存放區屬性都可供使用。若要檢視關聯的映像集、匯入和標籤，請選擇適用的標籤。

### AWS CLI 和 SDKs

#### Bash

##### AWS CLI 使用 Bash 指令碼

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

#####
# function imaging_get_datastore
#
# Get a data store's properties.
#
# Parameters:
#     -i data_store_id - The ID of the data store.
#
```

```
# Returns:
#     [datastore_name, datastore_id, datastore_status, datastore_arn,
#      created_at, updated_at]
# And:
#     0 - If successful.
#     1 - If it fails.
#####
function imaging_get_datastore() {
    local datastore_id option OPTARG # Required to use getopt command in a
    function.
    local error_code
    # bashsupport disable=BP5008
    function usage() {
        echo "function imaging_get_datastore"
        echo "Gets a data store's properties."
        echo "  -i datastore_id - The ID of the data store."
        echo ""
    }

    # Retrieve the calling parameters.
    while 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  
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [GetDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 取得資料存放區的屬性

下列get-datastore程式碼範例會取得資料存放區的屬性。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得資料存放區屬性](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[GetDatastore](#)。

## Java

### SDK for Java 2.x

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的[GetDatastore](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //         datastoreName: 'my_datastore',
    //         datastoreStatus: 'ACTIVE',
    //         updatedAt: 2023-08-04T18:50:36.239Z
    //     }
    // }
    return response.datastoreProperties;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [GetDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Python (Boto3) API 參考中的 [GetDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 列出資料存放區

使用 `ListDatastores` 動作列出 AWS HealthImaging 中的可用 [資料存放區](#)。下列功能表提供的程序 AWS Management Console，以及 AWS CLI 和 AWS SDKs 程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考 [ListDatastores](#) 中的。

### 列出資料存放區

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

- 開啟 HealthImaging 主控台 [資料存放區頁面](#)。

所有資料存放區都會列在資料存放區區段下。

## AWS CLI 和 SDKs

### Bash

#### AWS CLI 使用 Bash 指令碼

```
#####
# 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
                ;;
        esac
    done
}
```

```
\?)  
    echo "Invalid parameter"  
    usage  
    return 1  
    ;;  
esac  
done  
export OPTIND=1  
  
local response  
response=$(aws medical-imaging list-datastores \  
    --output text \  
    --query "datastoreSummaries[*][datastoreName, datastoreId, datastoreStatus]")  
error_code=${?}  
  
if [[ $error_code -ne 0 ]]; then  
    aws_cli_error_log $error_code  
    errecho "ERROR: AWS reports list-datastores operation failed.$response"  
    return 1  
fi  
  
echo "$response"  
  
return 0  
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 列出資料存放區

下列list-datastores程式碼範例會列出可用的資料存放區。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[列出資料存放](#)區。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[ListDatastores](#)。

## Java

### SDK for Java 2.x

```
public static List<DatastoreSummary>  
listMedicalImagingDatastores(MedicalImagingClient medicalImagingClient) {  
    try {  
        ListDatastoresRequest datastoreRequest =  
ListDatastoresRequest.builder()  
            .build();  
        ListDatastoresIterable responses =  
medicalImagingClient.listDatastoresPaginator(datastoreRequest);  
        List<DatastoreSummary> datastoreSummaries = new ArrayList<>();  
  
        responses.stream().forEach(response ->  
datastoreSummaries.addAll(response.datastoreSummaries()));  
  
        return datastoreSummaries;  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
    }  
}
```

```
        System.exit(1);
    }

    return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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 it
        // 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:datasotre/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     datastoreName: 'my_datastore',
//     datastoreStatus: 'ACTIVE',
//     updatedAt: 2023-08-04T18:49:54.429Z
//   }
//   ...
// ]
// }

return datastoreSummaries;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

    :return: The list of data stores.
    """

    try:
        paginator =
self.health_imaging_client.getPaginator("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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Python (Boto3) API 參考中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

### ⓘ 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 刪除資料存放區

使用 `DeleteDatastore` 動作來刪除 AWS HealthImaging [資料存放區](#)。下列功能表提供的程序 AWS Management Console，以及 AWS CLI 和 AWS SDKs 程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考[`DeleteDatastore`](#)中的。

### ⓘ Note

您必須先刪除其中的所有[映像集](#)，才能刪除資料存放區。如需詳細資訊，請參閱[刪除映像集](#)。

### 刪除資料存放區

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。
2. 選擇資料存放區。
3. 選擇刪除。

刪除資料存放區頁面隨即開啟。

4. 若要確認刪除資料存放區，請在文字輸入欄位中輸入資料存放區名稱。
5. 選擇刪除資料存放區。

### AWS CLI 和 SDKs

#### Bash

##### AWS CLI 使用 Bash 指令碼

```
#####
# function errecho
#
```

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

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

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

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

```
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
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [DeleteDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 刪除資料存放區

下列delete-datastore程式碼範例會刪除資料存放區。

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

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[刪除資料存放區](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[DeleteDatastore](#)。

## Java

### SDK for Java 2.x

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的[DeleteDatastore](#)。

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[AWS 程式碼範例儲存庫](#)中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     datastoreStatus: 'DELETING'
    // }

    return response;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [DeleteDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [DeleteDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

### ⓘ 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 了解儲存層

AWS HealthImaging 使用智慧型分層進行自動臨床生命週期管理。這會導致新的或作用中資料和長期封存資料具有令人信服的效能和價格，而不會產生摩擦。HealthImaging 使用下列方案，每月每 GB 計費儲存。

- 經常存取層級 – 經常存取資料的層級。
- 封存即時存取層 – 封存資料的層。

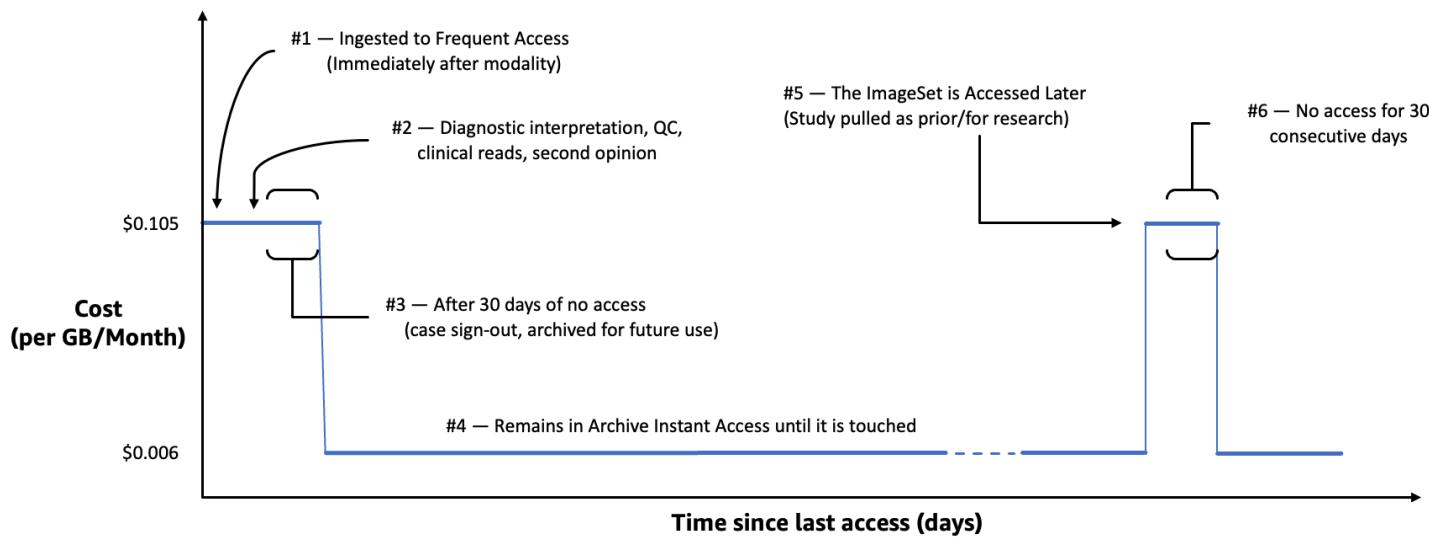
### ⓘ Note

常用存取和封存即時存取層之間沒有效能差異。智慧分層會套用至特定[映像集](#) API 動作。智慧分層無法辨識資料存放區、匯入和標記 API 動作。層之間的移動會根據 API 用量自動進行，並會在下一節中說明。

### 層移動如何運作？

- 匯入後，影像集會以常用存取層啟動。
- 連續 30 天未觸控後，影像集會自動移至封存即時存取層。
- 封存即時存取層中的影像集只會在碰觸之後移回常用存取層。

下圖提供 HealthImaging 智慧分層程序的概觀。



## 什麼是觸控？

觸控是透過 AWS Management Console、或 AWS SDKs 的特定 API 存取 AWS CLI，並會在下列情況發生：

1. 建立新的映像集 (StartDICOMImportJob 或 CopyImageSet)
2. 映像集已更新 (UpdateImageSetMetadata 或 CopyImageSet)
3. 會讀取影像集的相關聯中繼資料或影像影格（像素資料）(GetImageSetMetaData 或 GetImageFrame)

下列 HealthImaging API 動作會導致碰觸並將映像集從封存即時存取層移至經常存取層。

- StartDICOMImportJob
- GetImageSetMetadata
- GetImageFrame
- CopyImageSet
- UpdateImageSetMetadata

### Note

雖然無法使用 UpdateImageSetMetadata 動作刪除 影像影格（像素資料），但仍會將其計入計費用途。

下列 HealthImaging API 動作不會導致接觸。因此，它們不會將映像集從封存即時存取層移至經常存取層。

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

# 使用 AWS HealthImaging 匯入影像資料

匯入是將醫療影像資料從 Amazon S3 輸入儲存貯體移至 AWS HealthImaging [資料存放區](#)的程序。在匯入期間，AWS HealthImaging 會先執行[像素資料驗證檢查](#)，再將您的 DICOM P10 檔案轉換為包含[中繼資料](#)和[影像影格](#)的[影像集](#)（像素資料）。

## 重要

HealthImaging 匯入任務會處理 DICOM 執行個體二進位檔 (.dcm 檔案)，並將其轉換為映像集。使用 HealthImaging [雲端原生動作 APIs](#) 來管理資料存放區和映像集。使用 HealthImaging [的 DICOMweb 服務表示法](#)來傳回 DICOMweb 回應。

下列主題說明如何使用 AWS CLI AWS Management Console 和 AWS SDKs 將醫療影像資料匯入 HealthImaging 資料存放區。

## 主題

- [了解匯入任務](#)
- [啟動匯入任務](#)
- [取得匯入任務屬性](#)
- [列出匯入任務](#)

## 了解匯入任務

在 AWS HealthImaging 中建立[資料存放區](#)之後，您必須將醫療影像資料從 Amazon S3 輸入儲存貯體匯入資料存放區，才能建立[影像集](#)。您可以使用 AWS CLI AWS Management Console 和 AWS SDKs 來啟動、描述和列出匯入任務。

當您將 DICOM P10 資料匯入 AWS HealthImaging 資料存放區時，服務會嘗試根據研究 UID、序列 UID、執行個體 UID 的 DICOM 階層，根據[中繼資料元素](#)自動組織執行個體。如果匯入資料的[中繼資料元素](#)與資料存放區中的現有主要[影像集](#)沒有衝突，則匯入的資料將成為主要資料。如果新匯入的 DICOM P10 資料的中繼資料元素與現有的主要[影像集](#)衝突，則新資料將新增至非主要[影像集](#)。

當資料匯入建立非主要[映像集](#)時，AWS HealthImaging 會使用 發出 EventBridge 事件 `isPrimary: False`，而寫入的記錄 `success.ndjson` 也會在 `importResponse` 物件 `isPrimary: False` 中。

當您匯入資料時，HealthImaging 會執行下列動作：

- 如果在一個匯入任務中匯入包含 DICOM 序列的執行個體，且執行個體未與資料存放區中已存在的執行個體衝突，則所有執行個體都會組織成一個主要映像集。
- 如果在兩個或多個匯入任務中匯入包含 DICOM 序列的執行個體，且執行個體未與資料存放區中已存在的執行個體衝突，則所有執行個體都會組織為一個主要映像集。
- 如果匯入執行個體超過一次，則最新版本會覆寫存放在主要映像集中的任何較舊版本，且主要映像集的版本編號會遞增。

您可以使用更新映像集中繼資料中所述的步驟來更新主要 中的執行個體。

從 Amazon S3 將醫療影像檔案匯入 HealthImaging 資料存放區時，請注意下列事項：

- 對應至 DICOM 系列的執行個體會自動合併在單一影像集中，表示主要影像集。
- 您可以在一個匯入任務或多個匯入任務中匯入 DICOM P10 資料，而服務會將執行個體整理成對應至 DICOM 系列的主要映像集
- 長度限制條件適用於匯入期間的特定 DICOM 元素。為了確保成功匯入任務，請確認您的醫學影像資料未超過長度限制。如需詳細資訊，請參閱[DICOM 元素限制條件](#)。
- 像素資料驗證檢查會在匯入任務開始時執行。如需詳細資訊，請參閱[像素資料驗證](#)。
- 有與 HealthImaging 匯入動作相關聯的端點、配額和限流限制。如需詳細資訊，請參閱[端點和配額及調節限制](#)。
- 對於每個匯入任務，處理結果會存放在 outputS3Uri 位置。處理結果會組織為 job-output-manifest.json 檔案和 SUCCESSFAILURE 資料夾。

 Note

單一匯入任務最多可包含 10,000 個巢狀資料夾。

- job-output-manifest.json 檔案包含已處理資料的 jobSummary 輸出和其他詳細資訊。下列範例顯示 檔案的輸出 job-output-manifest.json。

```
{  
  "jobSummary": {  
    "jobId": "09876543210987654321098765432109",  
    "datastoreId": "12345678901234567890123456789012",  
    "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",  
    "outputS3Uri": "s3://medical-imaging-dicom-output/dicom_output/",  
    "status": "PENDING",  
    "statusReason": null,  
    "lastModified": "2023-01-12T10:00:00Z",  
    "lastUpdated": "2023-01-12T10:00:00Z",  
    "processingTime": 0  
  }  
}
```

```

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

```

- SUCCESS 資料夾會保留 `success.ndjson` 檔案，其中包含成功匯入的所有影像檔案的結果。下列範例顯示 檔案的輸出 `success.ndjson`。

```

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

```

- FAILURE 資料夾會保留 `failure.ndjson` 檔案，其中包含未成功匯入的所有影像檔案的結果。下列範例顯示 檔案的輸出 `failure.ndjson`。

```

{"inputFile":"dicom_input/invalidDicomFile1.dcm","exception": {"exceptionType":"ValidationException","message":"DICOM attribute TransferSyntaxUID does not exist"}}

```

```
{"inputFile":"dicom_input/invalidDicomFile2.dcm","exception": {"exceptionType":"ValidationException","message":"DICOM attributes does not exist"}}
```

- 匯入任務會保留在任務清單中 90 天，然後封存。

## 啟動匯入任務

使用 `StartDICOMImportJob` 動作啟動 [像素資料驗證檢查](#)，並將大量資料匯入 AWS HealthImaging [資料存放區](#)。匯入任務會匯入位於 `inputS3Uri` 參數所指定 Amazon S3 輸入儲存貯體中的 DICOM P10 檔案。匯入任務處理結果會存放在 `outputS3Uri` 參數指定的 Amazon S3 輸出儲存貯體中。

### Note

在開始匯入任務之前，請謹記下列事項：

- HealthImaging 支援匯入具有不同傳輸語法的 DICOM P10 檔案。有些檔案會在匯入期間保留其原始傳輸語法編碼，而其他檔案預設會轉碼為 HTJ2K 無失真。如需詳細資訊，請參閱[支援的傳輸語法](#)。
- HealthImaging 支援從位於其他[支援區域的](#) Amazon S3 儲存貯體匯入資料。若要實現此功能，請在啟動匯入任務時提供 `inputOwnerAccountId` 參數。如需詳細資訊，請參閱[的跨帳戶匯入 AWS HealthImaging](#)。
- HealthImaging 會在匯入期間將長度限制套用至特定 DICOM 元素。如需詳細資訊，請參閱[DICOM 元素限制條件](#)。

下列功能表提供 AWS Management Console 和 AWS SDKs 程式碼範例 AWS CLI 的程序。如需詳細資訊，請參閱 AWS HealthImaging API 參考[StartDICOMImportJob](#) 中的。

### 啟動匯入任務

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

- 開啟 HealthImaging 主控台[資料存放區頁面](#)。
- 選擇資料存放區。
- 選擇匯入 DICOM 資料。

匯入 DICOM 資料頁面隨即開啟。

4. 在詳細資訊區段下，輸入下列資訊：

- 名稱（選用）
  - 在 S3 中匯入來源位置
  - 來源儲存貯體擁有者的帳戶 ID（選用）
  - 加密金鑰（選用）
  - S3 中的輸出目的地
5. 在服務存取區段下，選擇使用現有的服務角色，然後從服務角色名稱功能表中選取角色，或選擇建立並使用新的服務角色。
6. 選擇匯入。

## AWS CLI 和 SDKs

### C++

#### SDK for C++

```
#!/ Routine which starts a HealthImaging import job.  
/*!  
 \param dataStoreID: The HealthImaging data store ID.  
 \param inputBucketName: The name of the Amazon S3 bucket containing the DICOM  
 files.  
 \param inputDirectory: The directory in the S3 bucket containing the DICOM  
 files.  
 \param outputBucketName: The name of the S3 bucket for the output.  
 \param outputDirectory: The directory in the S3 bucket to store the output.  
 \param roleArn: The ARN of the IAM role with permissions for the import.  
 \param importJobId: A string to receive the import job ID.  
 \param clientConfig: Aws client configuration.  
 \return bool: Function succeeded.  
 */  
bool AwsDoc::Medical_Imaging::startDICOMImportJob(
```

```
const Aws::String &dataStoreID, const Aws::String &inputBucketName,
const Aws::String &inputDirectory, const Aws::String &outputBucketName,
const Aws::String &outputDirectory, const Aws::String &roleArn,
Aws::String &importJobId,
const Aws::Client::ClientConfiguration &clientConfig) {
Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(clientConfig);
Aws::String inputURI = "s3://" + inputBucketName + "/" + inputDirectory +
"/";
Aws::String outputURI = "s3://" + outputBucketName + "/" + outputDirectory +
"/";
Aws::MedicalImaging::Model::StartDICOMImportJobRequest
startDICOMImportJobRequest;
startDICOMImportJobRequest.SetDatastoreId(dataStoreID);
startDICOMImportJobRequest.SetDataAccessRoleArn(roleArn);
startDICOMImportJobRequest.SetInputS3Uri(inputURI);
startDICOMImportJobRequest.SetOutputS3Uri(outputURI);

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

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

return startDICOMImportJobOutcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱 適用於 C++ 的 AWS SDK API 參考中的 [StartDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 啟動 dicom 匯入任務

下列start-dicom-import-job程式碼範例會啟動 dicom 匯入任務。

```
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"
```

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[啟動匯入任務](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的 [StartDICOMImportJob](#)。

## Java

### SDK for Java 2.x

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

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

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

    return "";
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [StartDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

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

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [StartDICOMImportJob](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

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

```
        err.response["Error"]["Message"],
    )
raise
else:
    return job["jobId"]
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [StartDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 取得匯入任務屬性

使用 GetDICOMImportJob 動作來進一步了解 AWS HealthImaging 匯入任務屬性。例如，在開始匯入任務之後，您可以執行 GetDICOMImportJob 來尋找任務的狀態。一旦 jobStatus 傳回為 COMPLETED，您就可以存取您的 [映像集](#)。

 Note

jobStatus 是指匯入任務的執行。因此，jobStatus COMPLETED 即使匯入程序期間發現驗證問題，匯入任務也可以傳回。如果 jobStatus 傳回為 COMPLETED，仍建議您檢閱寫入 Amazon S3 的輸出資訊清單，因為它們提供有關個別 P10 物件匯入成功或失敗的詳細資訊。

下列功能表提供 AWS Management Console 和 AWS SDKs 程式碼範例 AWS CLI 的程序。如需詳細資訊，請參閱 AWS HealthImaging API 參考[GetDICOMImportJob](#)中的。

## 若要取得匯入任務屬性

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

## AWS 主控台

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。

2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟。預設會選取影像集索引標籤。

3. 選擇匯入索引標籤。

4. 選擇匯入任務。

匯入任務詳細資訊頁面會開啟，並顯示匯入任務的相關屬性。

## 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;
}
```

- 如需 API 詳細資訊，請參閱 適用於 C++ 的 AWS SDK API 參考中的 [GetDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

若要取得 dicom 匯入任務的屬性

下列get-dicom-import-job程式碼範例會取得 dicom 匯入任務的屬性。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得匯入任務屬性](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的 [GetDICOMImportJob](#)。

## Java

### SDK for Java 2.x

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

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

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

    return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [GetDICOMImportJob](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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, 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     endedAt: 2023-09-19T17:29:21.753Z,
  //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobName: 'job_1',
  //     jobStatus: 'COMPLETED'
  //   }
}
```

```
//           outputS3Uri: 's3://health-imaging-dest/
output_ct/'xxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxx'',
//           submittedAt: 2023-09-19T17:27:25.143Z
//       }
// }

return response;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [GetDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param job_id: The ID of the job.
        :return: The job properties.
        """
        try:
            job = self.health_imaging_client.get_dicom_import_job(
                jobId=job_id, datastoreId=datastore_id
            )
        except ClientError as err:
            logger.error(
```

```
        "Couldn't get DICOM import job. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
raise
else:
    return job["jobProperties"]
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Python (Boto3) API 參考中的 [GetDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 列出匯入任務

使用 ListDICOMImportJobs 動作來列出針對特定 HealthImaging [資料存放區](#) 建立的匯入任務。下列功能表提供的程序 AWS Management Console，以及 AWS CLI 和 AWS SDKs 程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考 [ListDICOMImportJobs](#) 中的。

 Note

匯入任務會保留在任務清單中 90 天，然後封存。

## 列出匯入任務

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。
2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟。預設會選取影像集索引標籤。

3. 選擇匯入索引標籤，列出所有相關聯的匯入任務。

### AWS CLI 和 SDKs

#### CLI

##### AWS CLI

###### 列出 dicom 匯入任務

下列list-dicom-import-jobs程式碼範例列出 dicom 匯入任務。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [列出匯入任務](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的 [ListDICOMImportJobs](#)。

## Java

### SDK for Java 2.x

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

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [ListDICOMImportJobs](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

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

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

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

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

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

```
    return jobSummaries;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [ListDICOMImportJobs](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :return: The list of jobs.
        """
        try:
            paginator = self.health_imaging_client.getPaginator(
                "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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [ListDICOMImportJobs](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

# 使用 AWS HealthImaging 存取映像集

在 AWS HealthImaging 中存取醫療影像資料通常涉及搜尋具有唯一金鑰的影像集，以及取得相關聯的中繼資料和影像影格（像素資料）。

## 重要

在匯入期間，HealthImaging 會處理 DICOM 執行個體二進位檔 (.dcm 檔案)，並將其轉換為映像集。使用 HealthImaging 雲端原生動作 APIs 來管理資料存放區和映像集。使用 HealthImaging 的 DICOMweb 服務表示法來傳回 DICOMweb 回應。

下列主題說明如何在 AWS Management Console AWS CLI、和 AWS SDKs 中使用 HealthImaging 雲端原生動作來搜尋影像集，並取得其相關聯的屬性、中繼資料和影像影格。

## 主題

- [了解映像集](#)
- [搜尋映像集](#)
- [取得映像集屬性](#)
- [取得映像集中繼資料](#)
- [取得影像集像素資料](#)

## 了解映像集

影像集是 AWS 類似 DICOM 系列的，可作為 AWS HealthImaging 的基礎。當您將 DICOM 資料匯入 HealthImaging 時，會建立映像集。服務會根據研究、系列和執行個體的 DICOM 階層，嘗試組織匯入的 P10 資料。

影像集的推出原因如下：

- 透過靈活的 APIs 支援各種醫療影像工作流程（臨床和非臨床）。
- 提供一種機制，以長期存放和協調重複和不一致的資料。如果匯入的 P10 資料與存放區中已存在的主要影像集衝突，則會保留為非主要影像集。解決中繼資料衝突後，該資料可以成為主要資料。
- 僅將相關資料分組，將病患安全最大化。
- 鼓勵清理資料，以協助提高不一致的可見性。如需詳細資訊，請參閱[修改映像集](#)。

**i 重要**

在清理之前臨床使用 DICOM 資料可能會導致病患受傷。

下列功能表會進一步詳細說明影像集，並提供範例和圖表，協助您了解其在 HealthImaging 中的功能和用途。

## 什麼是影像集？

影像集是定義抽象分組機制的 AWS 概念，用於最佳化類似 DICOM 系列的相關醫療影像資料。當您將 DICOM P10 影像資料匯入 AWS HealthImaging 資料存放區時，它會轉換為包含中繼資料和影像影格的影像集（像素資料）。

**i Note**

影像集中繼資料已標準化。換言之，一組常見的屬性和值會映射至 DICOM 資料元素登錄檔中列出的病患、檢查和序列層級元素。HealthImaging 將傳入的 DICOM P10 物件分組為影像集時，會使用下列 DICOM 元素。

### 用於建立影像集的 DICOM 元素

元素名稱	元素標籤
<b>研究層級元素</b>	
Study Date	(0008,0020)
Accession Number	(0008,0050)
Patient ID	(0010,0020)
Study Instance UID	(0020,000D)
Study ID	(0020,0010)
<b>系列層級元素</b>	
Series Instance UID	(0020,000E)

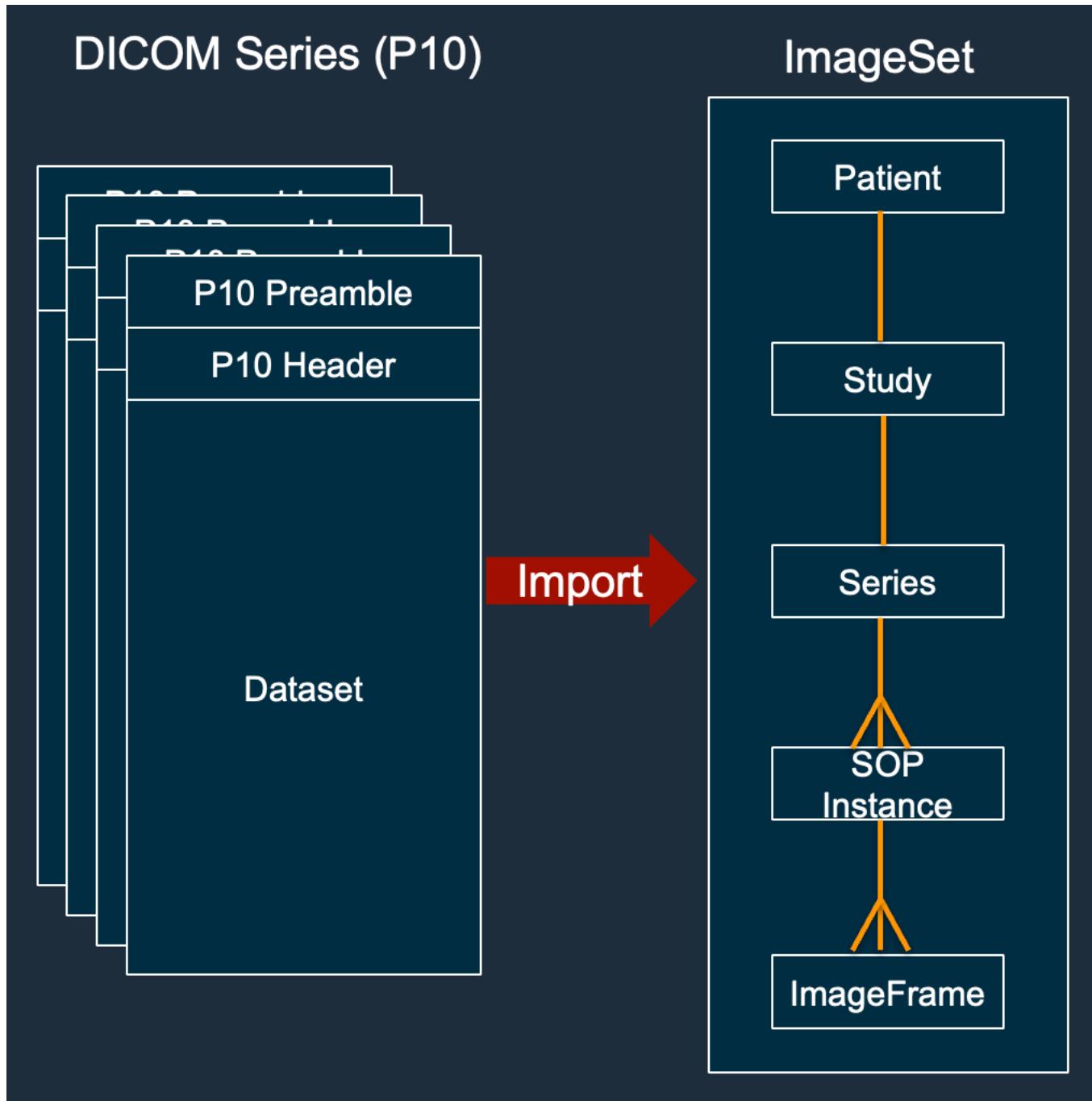
元素名稱	元素標籤
Series Number	(0020,0011)

在匯入期間，某些影像集會保留其原始傳輸語法編碼，而其他影像集則預設為無損轉碼為高輸送量 JPEG 2000 (HTJ2K)。如果影像集以 HTJ2K 編碼，則必須在檢視之前解碼。如需詳細資訊，請參閱支援的傳輸語法及 HTJ2K 解碼程式庫。

影像影格（像素資料）以高輸送量 JPEG 2000 (HTJ2K) 編碼，且必須在檢視之前解碼。

影像集是 AWS 資源，因此會獲指派 [Amazon Resource Name \(ARNs\)](#)。它們可以標記最多 50 個鍵值對，並透過 IAM 授予[角色型存取控制 \(RBAC\)](#) 和[屬性型存取控制 \(ABAC\)](#)。此外，映像集會[進行版本](#)控制，因此會保留所有變更，並可存取先前的版本。

匯入 DICOM P10 資料會導致影像集中包含相同 DICOM 系列中一或多個 Service-Object 配對 (SOP) 執行個體的 DICOM 中繼資料和影像影格。



i Note

DICOM 匯入任務：

- 一律建立新的映像集或增加現有映像集的版本。
- 請勿刪除重複的 SOP 執行個體儲存體。相同 SOP 執行個體的每個匯入都會使用額外的儲存體做為新的非主要映像集，或現有主要映像集的遞增版本。

- 自動組織具有一致、不衝突中繼資料的 SOP 執行個體做為主要影像集，其中包含具有一致病患、檢查和序列中繼資料元素的執行個體。
- 如果在兩個或多個匯入任務中匯入包含 DICOM 序列的執行個體，且執行個體不會與資料存放區中已存在的執行個體衝突，則所有執行個體都會組織在一個主要映像集中。
- 建立包含 DICOM P10 資料的非主要影像集，該資料集與資料存放區中已存在的主要影像集衝突。
- 將最近收到的資料保留為主要影像集的最新版本。
  - 如果包含 DICOM 序列的執行個體是主要影像集，且再次匯入一個執行個體，則新副本將插入主要影像集，且版本將遞增。

## 影像集中繼資料是什麼樣子？

使用 `GetImageSetMetadata` 動作來擷取影像集中繼資料。傳回的中繼資料會以 **壓縮 gzip**，因此您必須在檢視之前將其解壓縮。如需詳細資訊，請參閱[取得映像集中繼資料](#)。

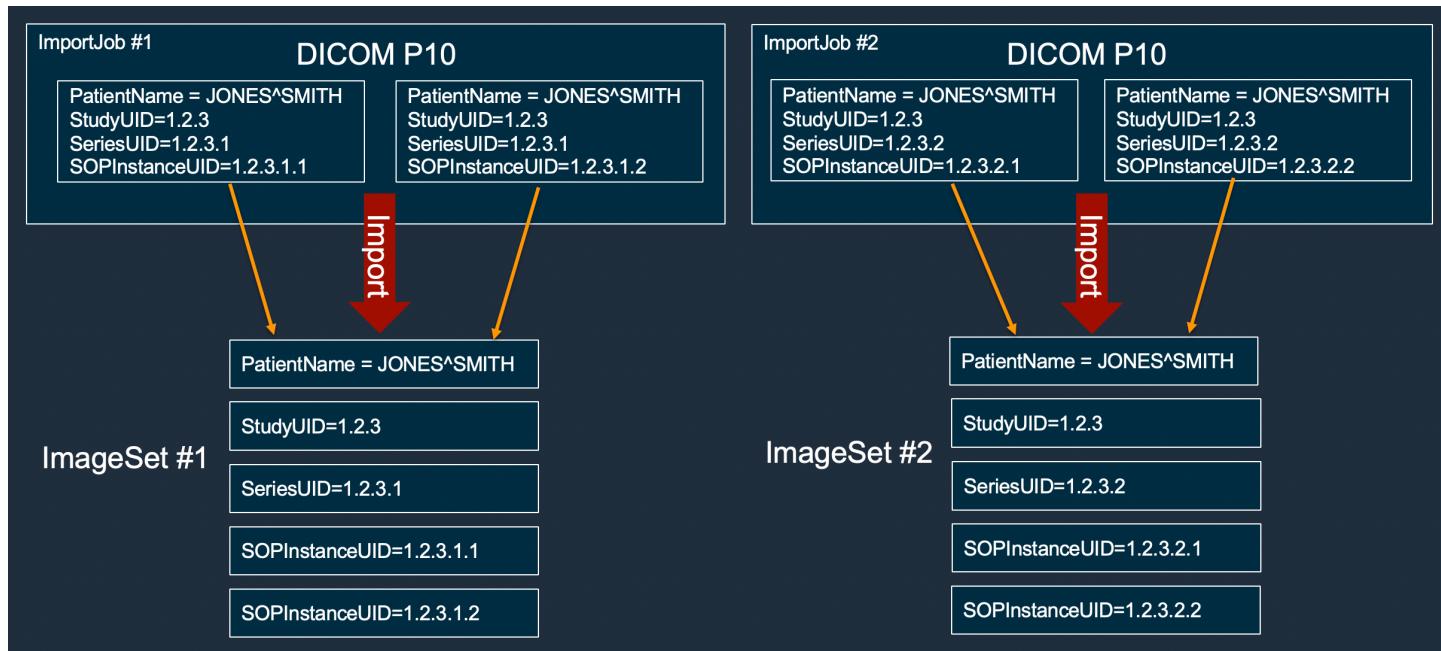
下列範例顯示 JSON 格式的影像集中繼資料結構。

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

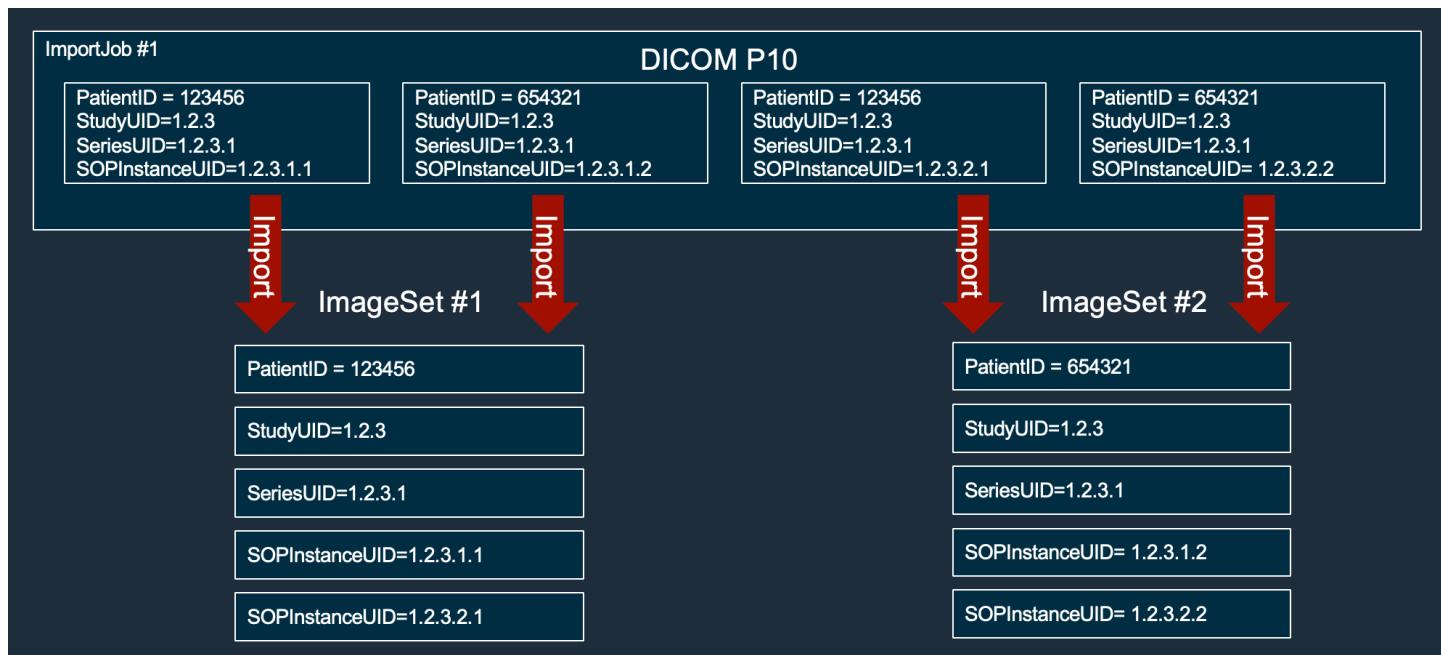
## 影像集建立範例：多個匯入任務

下列範例顯示多個匯入任務如何一律建立新的映像集，且絕不會新增至現有的映像集。



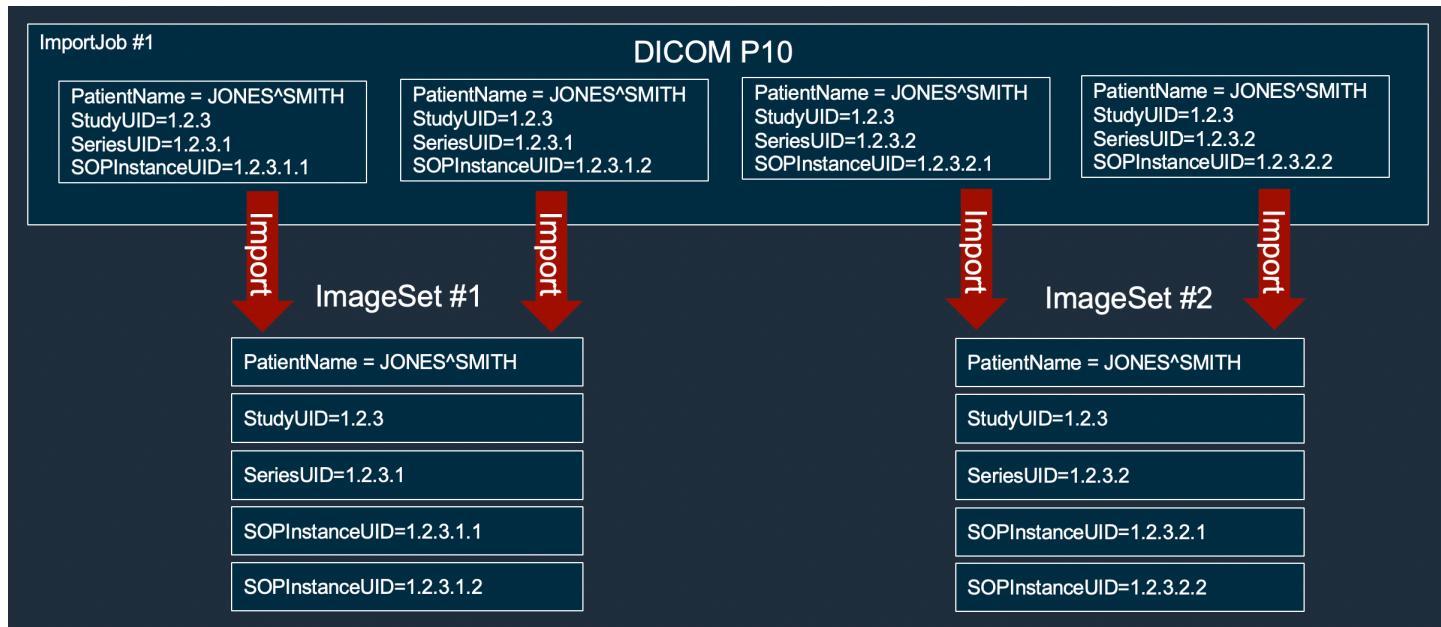
## 影像集建立範例：具有兩個變體的單一匯入任務

下列範例顯示無法合併至單一映像集的單一匯入任務，因為執行個體 1 和 3 具有與執行個體 2 和 4 不同的病患 IDs。若要解決此問題，您可以使用 `UpdateImageSetMetadata` 動作來解決病患 ID 與現有主要影像集的衝突。解決衝突後，您可以使用 `CopyImageSet` 動作搭配引數 `--promoteToPrimary`，將影像集新增至主要影像集。



## 影像集建立範例：具有最佳化的單一匯入任務

下列範例顯示單一匯入任務建立兩個影像集以改善輸送量，即使病患名稱相符。



## 搜尋映像集

使用 `SearchImageSets` 動作，針對 ACTIVE HealthImaging 資料存放區中的所有 [映像集](#) 執行搜尋查詢。下列功能表提供 AWS Management Console 和 程式碼範例的程序，適用於 AWS CLI 和 SDK。AWS SDKs 如需詳細資訊，請參閱 AWS HealthImaging API 參考 [SearchImageSets](#) 中的。

### Note

搜尋映像集時，請記住下列幾點。

- `SearchImageSets` 接受單一搜尋查詢參數，並傳回具有相符條件之所有影像集的分頁回應。所有日期範圍查詢都必須輸入為 (`lowerBound`, `upperBound`)。
- 根據預設，`SearchImageSets` 會使用 `updatedAt` 欄位，依從最新到最舊的順序進行排序。
- 如果您使用客戶擁有的 AWS KMS 金鑰建立資料存放區，您必須先更新 AWS KMS 金鑰政策，才能與映像集互動。如需詳細資訊，請參閱 [建立客戶受管金鑰](#)。

## 搜尋映像集

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

## AWS 主控台

### Note

下列程序說明如何使用 Series Instance UID 和 Updated at 屬性篩選條件來搜尋映像集。

### Series Instance UID

#### 使用 Series Instance UID 屬性篩選條件搜尋映像集

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。
2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟，且影像集索引標籤預設為選取。

3. 選擇屬性篩選條件選單，然後選擇 Series Instance UID。
4. 在輸入要搜尋的值欄位中，輸入（貼上）感興趣的系列執行個體 UID。

### Note

系列執行個體 UID 值必須與 [DICOM 唯一識別碼 \(UIDs\) 登錄](#)檔中列出的值相同。請注意，要求包括一系列數字，這些數字之間至少包含一個句點。系列執行個體 UIDs 開頭或結尾不允許使用期間。不允許使用字母和空格，因此複製和貼上 UIDs 時請務必小心。

5. 選擇日期範圍選單，選擇系列執行個體 UID 的日期範圍，然後選擇套用。
6. 選擇 Search (搜尋)。

根據預設，在所選日期範圍內的系列執行個體 UIDs 會以最新順序傳回。

### Updated at

#### 使用 Updated at 屬性篩選條件搜尋映像集

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。

## 2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟，影像集索引標籤預設為選取。

3. 選擇屬性篩選條件選單，然後選擇 Updated at。
4. 選擇日期範圍選單，選擇影像集日期範圍，然後選擇套用。
5. 選擇 Search (搜尋)。

根據預設，屬於所選日期範圍的影像集會以最新順序傳回。

## AWS CLI 和 SDKs

### C++

#### SDK for C++

用於搜尋映像集的公用程式函數。

```
///! 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 運算子。

```
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::Op
    .WithValues({Aws::MedicalImaging::Model::SearchByAttributeValue().WithDICOMPati
    });

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;
        }
    }
}
```

使用案例 #2：使用 DICOMStudyDate 和 DICOMStudyTime 的 BETWEEN 運算子。

```
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()).ToLocalTimeStamp
"%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 運算子。先前保留研究的時間。

```
Aws::MedicalImaging::Model::SearchByAttributeValue useCase3StartDate;

useCase3StartDate.SetCreatedAt(Aws::Utils::DateTime("20231130T00000000Z", 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;
    }
}
```

```
}
```

使用案例 #4：DICOMSeriesInstanceUID 上的 EQUAL 運算子和 updatedAt 上的 BETWEEN，並在 updatedAt 欄位中依 ASC 順序排序回應。

```
Aws::MedicalImaging::Model::SearchByAttributeValue useCase4StartDate;  
useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T00000000Z", 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;
    }
}
```

- 如需 API 詳細資訊，請參閱 適用於 C++ 的 AWS SDK API 參考中的 [SearchImageSets](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 範例 1：使用 EQUAL 運算子搜尋映像集

下列search-image-sets程式碼範例使用 EQUAL 運算子，根據特定值搜尋影像集。

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

#### search-criteria.json 的內容

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

輸出：

```
{  
    "imageSetsMetadataSummaries": [  
        {"imageSetId": "09876543210987654321098765432109",  
         "createdAt": "2022-12-06T21:40:59.429000+00:00",  
         "version": 1,  
         "DICOMTags": {  
             "DICOMStudyId": "2011201407",  
             "DICOMStudyDate": "19991122",  
             "DICOMPatientSex": "F",  
             "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",  
             "DICOMPatientBirthDate": "19201120",  
             "DICOMStudyDescription": "UNKNOWN",  
             "DICOMPatientId": "SUBJECT08701",  
             "DICOMPatientName": "Melissa844 Huel628",  
             "DICOMNumberofStudyRelatedInstances": 1,  
             "DICOMStudyTime": "140728",  
             "DICOMNumberofStudyRelatedSeries": 1  
         },  
         "updatedAt": "2022-12-06T21:40:59.429000+00:00"  
     }]  
}
```

範例 2：使用 DICOMStudyDate 和 DICOMStudyTime 使用 BETWEEN 運算子搜尋影像集

下列search-image-sets程式碼範例會搜尋 1990 年 1 月 1 日至 2023 年 1 月 1 日 (12 : 00 AM) 之間產生 DICOM 檢查的影像集。

注意：DICOMStudyTime 是選用的。如果不存在，則上午 12 : 00 (一天的開始) 是提供用於篩選日期的時間值。

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

search-criteria.json 的內容

```
{  
    "filters": [{}  
    "values": [{}]
```

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

輸出：

```
{
    "imageSetsMetadataSummaries": [
        {
            "imageSetId": "09876543210987654321098765432109",
            "createdAt": "2022-12-06T21:40:59.429000+00:00",
            "version": 1,
            "DICOMTags": {
                "DICOMStudyId": "2011201407",
                "DICOMStudyDate": "19991122",
                "DICOMPatientSex": "F",
                "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",
                "DICOMPatientBirthDate": "19201120",
                "DICOMStudyDescription": "UNKNOWN",
                "DICOMPatientId": "SUBJECT08701",
                "DICOMPatientName": "Melissa844 Huel628",
                "DICOMNumberOfStudyRelatedInstances": 1,
                "DICOMStudyTime": "140728",
                "DICOMNumberOfStudyRelatedSeries": 1
            },
            "updatedAt": "2022-12-06T21:40:59.429000+00:00"
        }
    ]
}
```

範例 3：使用 createdAt 使用 BETWEEN 運算子搜尋映像集（先前保留研究的時間）

下列search-image-sets程式碼範例會搜尋 DICOM 檢查在 UTC 時區的時間範圍之間保留在 HealthImaging 中的影像集。

注意：以範例格式 ("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 的內容

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

輸出：

```
{
  "imageSetsMetadataSummaries": [
    {
      "imageSetId": "09876543210987654321098765432109",
      "createdAt": "2022-12-06T21:40:59.429000+00:00",
      "version": 1,
      "DICOMTags": {
        "DICOMStudyId": "2011201407",
        "DICOMStudyDate": "19991122",
        "DICOMPatientSex": "F",
        "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",
        "DICOMPatientBirthDate": "19201120",
        "DICOMStudyDescription": "UNKNOWN",
        "DICOMPatientId": "SUBJECT08701",
        "DICOMPatientName": "Melissa844 Huel628",
        "DICOMNumberOfStudyRelatedInstances": 1,
        "DICOMStudyTime": "140728",
        "DICOMNumberOfStudyRelatedSeries": 1
      },
      "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }
]
```

```
}
```

範例 4：使用 DICOMSeriesInstanceUID 上的 EQUAL 運算子搜尋影像集，並在 updatedAt 上依 ASC 順序排序回應 updatedAt

下列search-image-sets程式碼範例會在 DICOMSeriesInstanceUID 上搜尋具有 EQUAL 運算子的映像集，並在 updatedAt 欄位中以 ASC 順序排序回應 updatedAt。

注意：以範例格式 ("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 的內容

```
{
  "filters": [
    {
      "values": [
        {
          "updatedAt": "2024-03-11T15:00:05.074000-07:00"
        },
        {
          "updatedAt": "2024-03-11T16:00:05.074000-07:00"
        }
      ],
      "operator": "BETWEEN"
    },
    {
      "values": [
        {
          "DICOMSeriesInstanceUID": "1.2.840.9999999.84710745.943275268089"
        }
      ],
      "operator": "EQUAL"
    }
  ],
  "sort": {
    "sortField": "updatedAt",
    "sortOrder": "ASC"
  }
}
```

輸出：

```
{
  "imageSetsMetadataSummaries": [
    {
      "imageSetId": "09876543210987654321098765432109",
      "createdAt": "2022-12-06T21:40:59.429000+00:00",
      "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }
  ]
}
```

```
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    ],
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [搜尋映像集](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的 [SearchImageSets](#)。

## Java

### SDK for Java 2.x

用於搜尋映像集的公用程式函數。

```
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 運算子。

```
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();
}
```

### 使用案例 #2 : 使用 DICOMStudyDate 和 DICOMStudyTime 的 BETWEEN 運算子。

```
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 運算子。先前保留研究的時間。

```
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();
    }
}
```

使用案例 #4 : DICOMSeriesInstanceUID 上的 EQUAL 運算子和 updatedAt 上的 BETWEEN ,  
並在 updatedAt 欄位中依 ASC 順序排序回應。

```
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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [SearchImageSets](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

用於搜尋映像集的公用程式函數。

```
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 運算子。

```
const datastoreId = "12345678901234567890123456789012";

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

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

## 使用案例 #2 : 使用 DICOMStudyDate 和 DICOMStudyTime 的 BETWEEN 運算子。

```
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` 運算子。先前保留研究的時間。

```
const datastoreId = "12345678901234567890123456789012";

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

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

使用案例 #4：DICOMSeriesInstanceUID 上的 `EQUAL` 運算子和 `updatedAt` 上的 `BETWEEN`，並在 `updatedAt` 欄位中依 ASC 順序排序回應。

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [

```

```
        { updatedAt: new Date("1985-04-12T23:20:50.52Z") },
        { updatedAt: new Date() },
    ],
    operator: "BETWEEN",
},
{
    values: [
    {
        DICOMSeriesInstanceUID:
        "1.1.123.123456.1.12.1.1234567890.1234.12345678.123",
    },
    ],
    operator: "EQUAL",
},
],
sort: {
    sortOrder: "ASC",
    sortField: "updatedAt",
},
};

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

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [SearchImageSets](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

用於搜尋映像集的公用程式函數。

```
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":  
[{"DICOPatientId": "3524578"}]}]}  
        :return: The list of image sets.  
        """  
        try:  
            paginator =  
self.health_imaging_client.getPaginator("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 運算子。

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

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

使用案例 #2：使用 DICOMStudyDate 和 DICOMStudyTime 的 BETWEEN 運算子。

```
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 運算子。先前保留研究的時間。

```
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}"
)
```

使用案例 #4：DICOMSeriesInstanceUID 上的 EQUAL 運算子和 updatedAt 上的 BETWEEN，並在 updatedAt 欄位中依 ASC 順序排序回應。

```
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{n{image_sets}}")
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [SearchImageSets](#)。

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

#### 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 取得映像集屬性

使用 GetImageSet 動作來傳回 HealthImaging 中指定 [影像集](#) 的屬性。下列功能表提供 的程序 AWS Management Console，以及 AWS CLI 和 AWS SDKs 程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考 [GetImageSet](#) 中的。

### Note

根據預設，AWS HealthImaging 會傳回映像集最新版本的屬性。若要檢視舊版映像集的屬性，請將您的請求versionId提供給。

使用 GetDICOMInstance HealthImaging 的 DICOMweb 服務表示法，傳回 DICOM 執行個體二進位檔 (.dcm 檔案)。如需詳細資訊，請參閱[從 HealthImaging 取得 DICOM 執行個體](#)。

## 取得影像集屬性

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。

2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟，影像集索引標籤預設為選取。

3. 選擇影像集。

影像集詳細資訊頁面會開啟並顯示影像集屬性。

## AWS CLI 和 SDKs

### CLI

#### AWS CLI

##### 取得影像集屬性

下列get-image-set程式碼範例會取得映像集的屬性。

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

輸出：

```
{
```

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得映像集屬性](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的[GetImageSet](#)。

## Java

### SDK for Java 2.x

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

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

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

    return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [GetImageSet](#)。

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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: 'xxxxxxxxxxxxxxxxxx',
//      imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxxxx: datastore/
xxxxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxx',
//      imagesetId: 'xxxxxxxxxxxxxxxxxx',
//      imagesetState: 'ACTIVE',
//      imageSetWorkflowStatus: 'CREATED',
//      updatedAt: 2023-09-22T14:49:26.427Z,
//      versionId: '1'
// }

return response;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [GetImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """

```

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [GetImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 取得映像集中繼資料

使用 `GetImageSetMetadata` 動作來擷取 HealthImaging 中指定影像集的中繼資料。下列功能表提供 和 AWS CLI AWS SDKs AWS Management Console 程式碼範例的程序。如需詳細資訊，請參閱 AWS HealthImaging API 參考 [GetImageSetMetadata](#) 中的。

### Note

根據預設，HealthImaging 會傳回影像集最新版本的中繼資料屬性。若要檢視舊版映像集的中繼資料，請將您的請求 `versionId` 提供給。

影像集中繼資料會以 壓縮，`gzip` 並以 JSON 物件的形式傳回。因此，您必須先解壓縮 JSON 物件，才能檢視標準化中繼資料。如需詳細資訊，請參閱 [中繼資料標準化](#)。

使用 `GetDICOMInstanceMetadata` HealthImaging 的 DICOMweb 服務表示法，傳回 DICOM 執行個體中繼資料 (.json 檔案)。如需詳細資訊，請參閱 [從 HealthImaging 取得 DICOM 執行個體中繼資料](#)。

### 若要取得影像集中繼資料

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

1. 開啟 HealthImaging 主控台 [資料存放區頁面](#)。
2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟，且影像集索引標籤預設為選取。

3. 選擇影像集。

影像集詳細資訊頁面隨即開啟，影像集中繼資料會顯示在影像集中繼資料檢視器區段下方。

### AWS CLI 和 SDKs

#### C++

##### SDK for C++

公用程式函數，以取得影像集中繼資料。

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

取得不含 版本的映像集中繼資料。

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

使用 版本取得映像集中繼資料。

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

- 如需 API 詳細資訊，請參閱 適用於 C++ 的 AWS SDK API 參考中的 [GetImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

範例 1：取得不含 版本的映像集中繼資料

下列get-image-set-metadata程式碼範例會取得映像集的中繼資料，而不需指定版本。

注意：outfile 是必要的參數

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

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

傳回的中繼資料會以 gzip 壓縮，並存放在 studymetadata.json.gz 檔案中。若要檢視傳回 JSON 物件的內容，您必須先解壓縮該物件。

輸出：

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

### 範例 2：使用 版本取得映像集中繼資料

下列get-image-set-metadata程式碼範例會取得具有指定版本之映像集的中繼資料。

注意：outfile 是必要的參數

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

傳回的中繼資料會以 gzip 壓縮，並存放在 studymetadata.json.gz 檔案中。若要檢視傳回 JSON 物件的內容，您必須先解壓縮該物件。

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得映像集中繼資料](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的 [GetImageSetMetadata](#)。

## Java

### SDK for Java 2.x

```
public static void getMedicalImageSetMetadata(MedicalImagingClient  
medicalImagingClient,  
        String destinationPath,  
        String datastoreId,  
        String imagesetId,  
        String versionId) {  
  
    try {  
        GetImageSetMetadataRequest.Builder getImageSetMetadataRequestBuilder  
= GetImageSetMetadataRequest.builder()  
            .datastoreId(datastoreId)  
            .imageSetId(imagesetId);  
  
        if (versionId != null) {  
            getImageSetMetadataRequestBuilder =  
getImageSetMetadataRequestBuilder.versionId(versionId);  
        }  
  
        medicalImagingClient.getImageSetMetadata(getImageSetMetadataRequestBuilder.build(),  
            FileSystems.getDefault().getPath(destinationPath));  
  
        System.out.println("Metadata downloaded to " + destinationPath);  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [GetImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

公用程式函數，以取得影像集中繼資料。

```
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;
};
```

取得不含 版本的映像集中繼資料。

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

使用 版本取得映像集中繼資料。

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

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [GetImageSetMetadata](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

公用程式函數，以取得影像集中繼資料。

```
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
```

取得不含 版本的映像集中繼資料。

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

使用 版本取得映像集中繼資料。

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [GetImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 取得影像集像素資料

影像影格是影像集中存在以組成 2D 醫療影像的像素資料。使用 `GetImageFrame`動作來擷取 HealthImaging 中指定影像集的 HTJ2K-encoded 影像影格。下列功能表提供 AWS CLI 和 AWS SDKs 程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考[GetImageFrame](#)中的。

 Note

使用 `GetImageFrame`動作時，請記住下列幾點：

- 在匯入期間，HealthImaging 會保留某些傳輸語法的編碼，但預設會將其他語法轉碼為 HTJ2K 無損。因此，必須先解碼影像影格，才能在影像檢視器中檢視。如需詳細資訊，請參閱支援的傳輸語法及 HTJ2K 解碼程式庫。
- 對於存放在 HealthImaging 中的執行個體，使用 MPEG 系列 Transfer Syntaxes (包括 MPEG2, MPEG-4 AVC/H.264 和 HEVC/H.265) 中編碼的一或多個影像影格，`GetImageFrame`動作會以儲存的 Transfer Syntax 傳回視訊物件。
- 根據預設，`GetImageFrame`動作會以執行個體的預存傳輸語法傳回影像影格。如需詳細資訊，請參閱支援的傳輸語法。
- 您也可以使用 `GetDICOMInstanceFrames` HealthImaging 的 DICOMweb 服務表示法，為與 DICOMweb 相容的檢視器和應用程式擷取 DICOM 執行個體影格 (multipart 請求)。如需詳細資訊，請參閱從 HealthImaging 取得 DICOM 執行個體影格。

## 取得影像集像素資料

根據您對 AWS HealthImaging 的存取偏好設定，選擇選單。

## AWS 主控台

### Note

影像影格必須以程式設計方式存取和解碼，因為 中無法使用影像檢視器 AWS Management Console。

如需解碼和檢視影像影格的詳細資訊，請參閱 [HTJ2K 解碼程式庫](#)。

## 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();
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

若要取得影像集像素資料

下列get-image-frame程式碼範例會取得影像影格。

```
aws medical-imaging get-image-frame \
--datastore-id "12345678901234567890123456789012" \
--image-set-id "98765412345612345678907890789012" \
```

```
--image-frame-information imageFrameId=3abf5d5d7ae72f80a0ec81b2c0de3ef4 \
imageframe.jph
```

注意：此程式碼範例不包含輸出，因為 GetImageFrame 動作會將像素資料串流傳回至 imageframe.jph 檔案。如需有關解碼和檢視影像影格的資訊，請參閱 HTJ2K 解碼程式庫。

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得影像集像素資料](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[GetImageFrame](#)。

## Java

### SDK for Java 2.x

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

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

        .imageFrameInformation(ImageFrameInformation.builder()

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

    medicalImagingClient.getImageFrame(getImageSetMetadataRequest,
FileSystems.getDefault().getPath(destinationPath));

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

```
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

        :param file_path_to_write: The path to write the image frame's HTJ2K
        encoded pixel data.
    
```

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

### 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

# 使用 AWS HealthImaging 修改映像集

DICOM 匯入任務通常會要求您修改[映像集](#)，原因如下：

- 病患安全
- 資料一致性
- 降低儲存成本

## 重要

在匯入期間，HealthImaging 會處理 DICOM 執行個體二進位檔 (.dcm 檔案)，並將其轉換為映像集。使用 HealthImaging [雲端原生動作 APIs](#) 來管理資料存放區和映像集。使用 HealthImaging [的 DICOMweb 服務表示法](#)來傳回 DICOMweb 回應。

HealthImaging 提供數個雲端原生 APIs以簡化映像集修改程序。下列主題說明如何使用 AWS CLI 和 AWS SDKs修改映像集。

## 主題

- [列出映像集版本](#)
- [更新影像集中繼資料](#)
- [複製映像集](#)
- [刪除映像集](#)

## 列出映像集版本

使用 `ListImageSetVersions`動作來列出 HealthImaging 中[影像集](#)的版本歷史記錄。下列功能表提供 的程序 AWS Management Console，以及 AWS CLI 和 AWS SDKs程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考[ListImageSetVersions](#)中的。

## Note

AWS HealthImaging 會記錄對映像集所做的每個變更。更新映像集中繼資料會在映像集歷史記錄中建立新的版本。如需詳細資訊，請參閱[更新影像集中繼資料](#)。

## 列出映像集的版本

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。
2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟，影像集索引標籤預設為選取。

3. 選擇影像集。

影像集詳細資訊頁面隨即開啟。

影像集版本會顯示在影像集詳細資訊區段下方。

### AWS CLI 和 SDKs

#### CLI

##### AWS CLI

##### 列出映像集版本

下列list-image-set-versions程式碼範例會列出映像集的版本歷史記錄。

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

輸出：

```
{
    "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  
}  
]  
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[列出映像集版本](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的[ListImageSetVersions](#)。

## Java

### SDK for Java 2.x

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

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

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

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

return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [ListImageSetVersions](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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 = "xxxxxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxxxx",
```

```
) => {
    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 it 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;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [ListImageSetVersions](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :return: The list of image set versions.
        """
        try:
            paginator = self.health_imaging_client.getPaginator(
                "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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [ListImageSetVersions](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 更新影像集中繼資料

使用 UpdateImageSetMetadata 動作來更新 AWS HealthImaging 中的映像集中繼資料。您可以使用此非同步程序來新增、更新和移除影像集中繼資料屬性，這些屬性是在匯入期間建立的 [DICOM 標準化元素](#) 資訊清單。您也可以使用 UpdateImageSetMetadata 動作移除系列和 SOP 執行個體，讓映像集與外部系統保持同步，並取消識別映像集中繼資料。如需詳細資訊，請參閱《AWS HealthImaging API 參考 [UpdateImageSetMetadata](#)》中的。

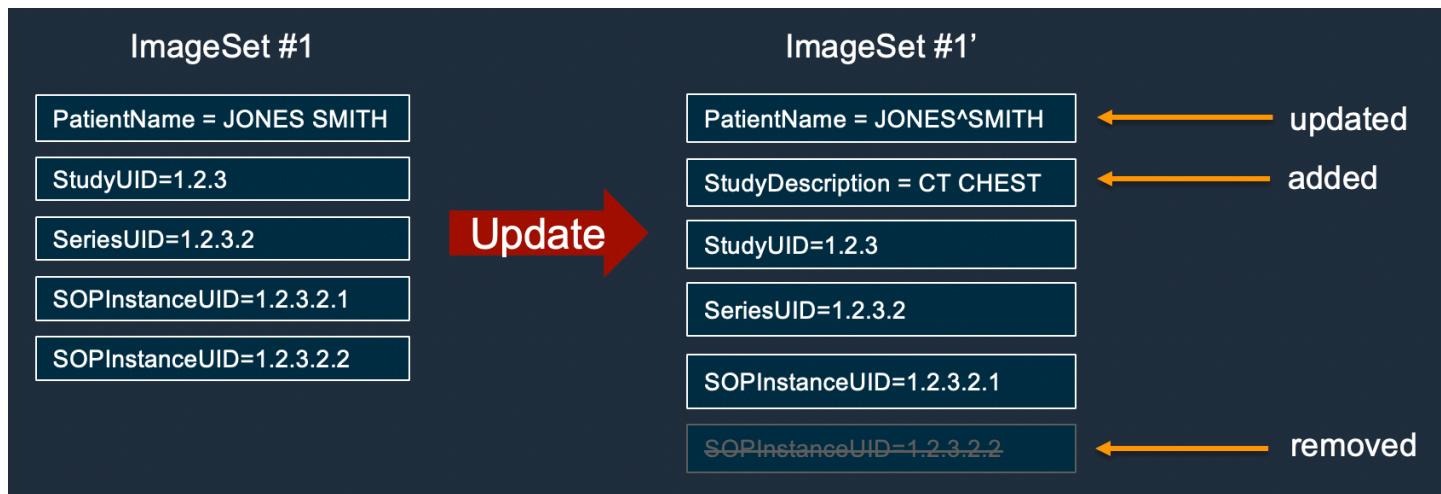
 Note

真實世界的 DICOM 匯入需要更新、新增和移除影像集中繼資料中的屬性。更新影像集中繼資料時，請謹記下列事項：

- 更新影像集中繼資料會在影像集歷史記錄中建立新的版本。如需詳細資訊，請參閱 [列出映像集版本](#)。若要還原至先前的映像集版本 ID，請使用選用 [revertToVersionId](#) 參數。

- 更新影像集中繼資料是非同步程序。因此，[imageSetState](#)和[imageSetWorkflowStatus](#)回應元素可用於提供正在進行更新之映像集的個別狀態和狀態。您無法對LOCKED影像集執行其他寫入操作。
- 如果UpdateImageSetMetadata動作不成功，請呼叫並檢閱[message](#)回應元素，以查看[common errors](#)。
- DICOM 元素限制條件會套用至中繼資料更新。[force](#) 請求參數可讓您在想要覆寫 的情況下，更新非主要[影像集](#)的元素[DICOM 中繼資料限制條件](#)。
- 無法更新主要[影像集](#)的患者和序列層級中繼資料元素。UpdateImageSet 不支援 --force 更新主要[影像集](#)的 StudyInstanceUID、SeriesInstanceUID 和 SOPInstanceUID。
- 設定[force](#)請求參數，以強制在非主要[映像集](#)上完成UpdateImageSetMetadata動作。設定此參數允許對映像集進行下列更新：
  - 更新 Tag.StudyInstanceUID、Tag.SOPInstanceUID、Tag.SeriesInstanceUID 和 Tag.StudyID 屬性
  - 新增、移除或更新執行個體層級私有 DICOM 資料元素
  - 將影像集提升為主要 的動作會變更影像集 ID。

下圖代表 HealthImaging 中正在更新的影像集中繼資料。



## 更新影像集中繼資料

根據您對 AWS HealthImaging 的存取偏好設定選擇索引標籤。

## AWS CLI 和 SDKs

### CLI

#### AWS CLI

##### 範例 1：在影像集中繼資料中插入或更新屬性

下列update-image-set-metadata範例會在影像集中繼資料中插入或更新屬性。

```
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 的內容

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

輸出：

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

##### 範例 2：從影像集中繼資料中移除屬性

下列update-image-set-metadata範例會從影像集中繼資料中移除屬性。

```
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 的內容

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

輸出：

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

範例 3：從映像集中繼資料中移除執行個體

下列update-image-set-metadata範例會從映像集中繼資料中移除執行個體。

```
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 的內容

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

輸出：

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

#### 範例 4：將映像集還原為先前的版本

下列update-image-set-metadata範例示範如何將映像集還原為舊版。CopyImageSet 和 UpdateImageSetMetadata 動作會建立新的映像集版本。

```
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"}'
```

輸出：

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

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

### 範例 5：將私有 DICOM 資料元素新增至執行個體

下列update-image-set-metadata範例顯示如何將私有元素新增至映像集中的指定執行個體。DICOM 標準允許私有資料元素進行無法包含在標準資料元素中的資訊通訊。您可以使用UpdateImageSetMetadata 動作建立、更新和刪除私有資料元素。

```
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 的內容

```
{  
    "DICOMUpdates": {  
        "updatableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series  
\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances  
\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\":  
\"001910F9\": \"97\"},\"DICOMVRs\": {\"001910F9\": \"DS\"}}}}}\"}  
    }  
}
```

輸出：

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

### 範例 6：將私有 DICOM 資料元素更新至執行個體

下列update-image-set-metadata範例顯示如何更新屬於映像集中執行個體之私有資料元素的值。

```
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 的內容

```
{
    "DICOMUpdates": {
        "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\": {\"00091001\": \"GE_GENESIS_DD\"}}}}}}}"
    }
}
```

輸出：

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

#### 範例 7：使用 強制參數更新 SOPInstanceUID

下列update-image-set-metadata範例示範如何使用 強制參數來覆寫 DICOM 中繼資料限制條件，以更新 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 的內容

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

### 輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[更新映像集中繼資料](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[UpdateImageSetMetadata](#)。

## Java

### SDK for Java 2.x

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

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

        UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

## 使用案例 #1：插入或更新屬性。

```
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 屬性。

```
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：移除執行個體。

```
final String removeInstance = """"  
{  
    "SchemaVersion": 1.1,  
    "Study": {  
        "Series": {  
  
            "1.1.1.1.1.12345.123456789012.123.12345678901234.1": {  
                "Instances": {}  
  
            "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：還原至先前的版本。

```
// 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);
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [UpdateImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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 = "xxxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxx",
    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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     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：插入或更新屬性並強制更新。

```
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 屬性。

```
// 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：移除執行個體。

```
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：還原至舊版。

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

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

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [UpdateImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

```
class MedicalImagingWrapper:  
    def __init__(self, health_imaging_client):  
        self.health_imaging_client = health_imaging_client  
  
    def update_image_set_metadata(  
        self, datastore_id, image_set_id, version_id, metadata, 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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

使用案例 #1：插入或更新屬性。

```
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：移除屬性。

```
# 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：移除執行個體。

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

"1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
            "Instances": {

"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：還原至舊版。

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

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [UpdateImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

您可以在映像集之間移動 SOP 執行個體、解決中繼資料元素衝突，以及使用 CopyImageSet、UpdateImageSetMetadata 和 DeleteImageSet APIs 從主要映像集新增或移除執行個體。

您可以使用 DeleteImageSet 動作從主要集合中移除映像集。

## 更新主要影像集的中繼資料

- 使用 CopyImageSet 動作建立非主要影像集，這是您要修改的主要影像集複本。假設這會傳回 103785414bc2c89330f7ce51bbd13f7a 為非主要影像集 ID。

```
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
```

- 使用 UpdateImageSetMetadata 動作對非主要影像集 進行變更 (103785414bc2c89330f7ce51bbd13f7a)。例如，變更 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": {
        "updateableAttributes": "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":{\"PatientID\":\"1234\"}}}"
    }
}'
```

### 3. 刪除您要修改的主要影像集。

```
aws medical-imaging delete-image-set --datastore-
    id a8d19e7875e1532d9b5652f6b25e12c9 --image-set-
    id 0778b83b36ec0b76752bfe32192fb7
```

### 4. 使用 CopyImageSet 動作搭配引數--promoteToPrimary，將更新的映像集新增至主要集合。

```
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. 刪除非主要影像集。

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

## 將非主要影像集設為主要

### 1. 使用 UpdateImageSetMetadata 動作來解決與現有主要映像集的衝突。

```
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": {  
        "updateableAttributes": "{\"SchemaVersion\":1.1,\"Patient\":{\"DICOM\":  
            {\"PatientID\":\"1234\"}}}"  
    }  
}'
```

2. 解決衝突時，請使用 CopyImageSet 動作搭配引數--promoteToPrimary，將影像集新增至主要影像集集合。

```
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. 確認 CopyImageSet 動作成功後，刪除來源非主要映像集。

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

## 複製映像集

使用 CopyImageSet 動作來複製 HealthImaging 中的 [映像集](#)。您可以使用此非同步程序，將映像集的內容複製到新的或現有的映像集。您可以將 複製到新的映像集以分割映像集，以及建立個別的複本。您也可以複製到現有的映像集，將兩個映像集合併在一起。如需詳細資訊，請參閱 AWS HealthImaging API 參考[CopyImageSet](#)中的。

### Note

使用 CopyImageSet 動作時，請記住下列事項：

- CopyImageSet 動作將建立新的映像集或新版本的 destinationImageSet。如需詳細資訊，請參閱[列出映像集版本](#)。
- Copy 是非同步程序。因此，狀態 ([imageSetState](#)) 和狀態 ([imageSetWorkflowStatus](#)) 回應元素可供您了解鎖定影像集上發生的操作。其他寫入操作無法在鎖定的影像集上執行。
- CopyImageSet 要求 SOP 執行個體 UIDs 映像集中是唯一的。

- 您可以使用 複製 SOP 執行個體的子集[copyableAttributes](#)。這可讓您從 挑選一或多個 SOP 執行個體sourceImageSet，以複製到 destinationImageSet。
- 如果CopyImageSet動作不成功，請呼叫 GetImageSet並檢閱 [message](#) 屬性。如需詳細資訊，請參閱[取得映像集屬性](#)。
- 真實世界的 DICOM 匯入可能會導致每個 DICOM 系列有多個影像集。除非提供選用[force](#)參數，否則CopyImageSet動作需要 sourceImageSet和 destinationImageSet 具有一致的中繼資料。
- 設定 [force](#) 參數以強制 操作，即使 sourceImageSet和 之間存在不一致的中繼資料元素destinationImageSet。在這些情況下，病患、檢查和序列中繼資料在 中保持不變destinationImageSet。

## 複製映像集

根據您對 AWS HealthImaging 的存取偏好設定選擇標籤。

### AWS CLI 和 SDKs

#### CLI

##### AWS CLI

範例 1：複製沒有目的地的影像集。

下列copy-image-set範例會複製沒有目的地的影像集。

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

輸出：

```
{  
    "destinationImageSetProperties": {  
        "latestVersionId": "2",  
        "imageSetWorkflowStatus": "COPYING",  
        "updatedAt": 1680042357.432,  
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",  
        "version": 2  
    },  
    "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",  
    "version": 2  
}
```

```
        "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"
}
```

## 範例 2：複製具有目的地的影像集。

下列copy-image-set範例會複製具有目的地的影像集複本。

```
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"} }'
```

輸出：

```
{
    "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"
}
```

範例 3：將來源映像集的執行個體子集複製到目的地映像集。

下列copy-image-set範例會將一個 DICOM 執行個體從來源映像集複製到目的地映像集。強制參數是用來覆寫病患、檢查和序列層級屬性中的不一致。

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

輸出：

```
{
  "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"
```

{}

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[複製映像集](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的[CopyImageSet](#)。

## Java

### SDK for Java 2.x

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

```
// Optionally copy a subset of image instances.  
if (subsets != null) {  
    String subsetInstanceToCopy =  
getCopiableAttributesJSON(imageSetId, subsets);  
  
copySourceImageSetInformation.dicomCopies(MetadataCopies.builder()  
    .copiableAttributes(subsetInstanceToCopy)  
    .build());  
}  
  
CopyImageSetInformation.Builder copyImageSetBuilder =  
CopyImageSetInformation.builder()  
    .sourceImageSet(copySourceImageSetInformation.build());  
  
// Optionally designate a destination image set.  
if (destinationImageSetId != null) {  
    copyImageSetBuilder =  
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()  
    .imageSetId(destinationImageSetId)  
    .latestVersionId(destinationVersionId)  
    .build());  
}  
  
CopyImageSetRequest copyImageSetRequest =  
CopyImageSetRequest.builder()  
    .datastoreId(datastoreId)  
    .sourceImageSetId(imageSetId)  
    .copyImageSetInformation(copyImageSetBuilder.build())  
    .force(force)  
    .build();  
  
CopyImageSetResponse response =  
medicalImagingClient.copyImageSet(copyImageSetRequest);  
  
    return response.destinationImageSetProperties().imageSetId();  
} catch (MedicalImagingException e) {  
    System.err.println(e.awsErrorDetails().errorMessage());  
    throw e;  
}  
}
```

用於建立可複製屬性的公用程式函數。

```
/*
 * 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [CopyImageSet](#)。

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

用於複製映像集的公用程式函數。

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

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

```
copyImageSetInformation: {
    sourceImageSet: { latestVersionId: sourceVersionId },
},
force: force,
};

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

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

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

params.copyImageSetInformation.dicomCopies = copySubsetsJson;
}

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

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

複製沒有目的地的影像集。

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

使用目的地複製映像集。

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

使用目的地複製影像集的子集，並強制複製。

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

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [CopyImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

用於複製映像集的公用程式函數。

```
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"].append(subset)
        response = self.health_imaging_client.copy_image_set(
            datastoreId=datastore_id,
            imageSetId=image_set_id,
            latestVersionId=version_id,
            destinationImageSetId=destination_image_set_id,
            destinationVersionId=destination_version_id,
            force=force,
            subsets=copySubsetsJson,
        )
        return response["copiedImageSetId"]
    except Exception as e:
        raise e
```

```
copySubsetsJson["Study"]["Series"]["imageSetId"]["Instances"]
[  
    subset
] = {}  
  
copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
    "copyableAttributes": 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"]
```

複製沒有目的地的影像集。

```
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,
)
```

使用目的地複製映像集。

```
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,
)
```

複製影像集的子集。

```
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,  
)
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [CopyImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 刪除映像集

使用 DeleteImageSet 動作來刪除 HealthImaging 中的 [映像集](#)。下列功能表提供 的程序 AWS Management Console，以及 AWS CLI 和 AWS SDKs 程式碼範例。如需詳細資訊，請參閱 AWS HealthImaging API 參考 [DeleteImageSet](#) 中的。

### 刪除映像集

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

- 開啟 HealthImaging 主控台 [資料存放區頁面](#)。
- 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟，影像集索引標籤預設為選取。

3. 選擇影像集，然後選擇刪除。

刪除影像集模態隨即開啟。

4. 提供影像集的 ID，然後選擇刪除影像集。

## 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();  
}
```

- 如需 API 詳細資訊，請參閱 適用於 C++ 的 AWS SDK API 參考中的 [DeleteImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 刪除映像集

下列delete-image-set程式碼範例會刪除映像集。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [刪除映像集](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考中的 [DeleteImageSet](#)。

## Java

### SDK for Java 2.x

```
public static void deleteMedicalImageSet(MedicalImagingClient  
medicalImagingClient,  
                                         String datastoreId,  
                                         String imagesetId) {
```

```
try {
    DeleteImageSetRequest deleteImageSetRequest =
DeleteImageSetRequest.builder()
    .datastoreId(datastoreId)
    .imagesetId(imagesetId)
    .build();

    medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [DeleteImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imagesetId - The image set ID.
 */
export const deleteImageSet = async (
    datastoreId = "xxxxxxxxxxxxxxxx",
    imagesetId = "xxxxxxxxxxxxxxxx",
) => {
    const response = await medicalImagingClient.send(
        new DeleteImageSetCommand({
```

```
        datastoreId: datastoreId,
        imageSetId: imageSetId,
    }),
);
console.log(response);
// {
//     '$metadata': {
//         httpStatusCode: 200,
//         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
//         extendedRequestId: undefined,
//         cfId: undefined,
//         attempts: 1,
//         totalRetryDelay: 0
//     },
//     datastoreId: 'xxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxx',
//     imageSetState: 'LOCKED',
//     imageSetWorkflowStatus: 'DELETING'
// }
return response;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [DeleteImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

    def delete_image_set(self, datastore_id, image_set_id):
```

```
"""
Delete an image set.

:param datastore_id: The ID of the data store.
:param image_set_id: The ID of the image set.
:return: The delete results.
"""

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [DeleteImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

# 使用 AWS HealthImaging 標記資源

您可以將中繼資料以標籤形式指派給 HealthImaging 資源 ([資料存放區](#)和[映像集](#))。每個標籤都是由使用者定義的金鑰和值組成的標籤。標籤可協助您管理、識別、組織、搜尋和篩選資源。

## ① 重要

請勿在標籤中存放受保護的健康資訊 (PHI)、個人身分識別資訊 (PII) 或其他機密或敏感資訊。  
標籤不適用於私人或敏感資料。

下列主題說明如何使用 AWS Management Console、AWS CLI 和 AWS SDKs 來使用 HealthImaging 標記操作。如需詳細資訊，請參閱 AWS 一般參考 指南中的[標記您的 AWS 資源](#)。

## 主題

- [標記資源](#)
- [列出資源的標籤](#)
- [取消標記資源](#)

## 標記資源

使用 [TagResource](#) 動作來標記 AWS HealthImaging 中的[資料存放區](#)和[映像集](#)。下列程式碼範例說明如何搭配 AWS Management Console AWS CLI、和 AWS SDKs 使用 TagResource 動作。如需詳細資訊，請參閱 AWS 一般參考 指南中的[標記您的 AWS 資源](#)。

## 標記資源

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

## AWS 主控台

1. 開啟 HealthImaging 主控台[資料存放區頁面](#)。
2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟。

3. 選擇詳細資訊索引標籤。

4. 在標籤區段下，選擇管理標籤。

隨即開啟管理標籤頁面。

5. 選擇 Add new tag (新增標籤)。

6. 輸入金鑰和值（選用）。

7. 選擇 Save changes (儲存變更)。

## AWS CLI 和 SDKs

### CLI

#### AWS CLI

##### 範例 1：標記資料存放區

下列tag-resource程式碼範例會標記資料存放區。

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

此命令不會產生輸出。

##### 範例 2：標記映像集

下列tag-resource程式碼範例會標記影像集。

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

此命令不會產生輸出。

如需詳細資訊，請參閱 [AWS HealthImaging 開發人員指南中的使用 HealthImaging 標記資源](#)。

AWS HealthImaging

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [TagResource](#)。

## Java

### SDK for Java 2.x

```
public static void tagMedicalImagingResource(MedicalImagingClient  
medicalImagingClient,  
        String resourceArn,  
        Map<String, String> tags) {  
    try {  
        TagResourceRequest tagResourceRequest = TagResourceRequest.builder()  
            .resourceArn(resourceArn)  
            .tags(tags)  
            .build();  
  
        medicalImagingClient.tagResource(tagResourceRequest);  
  
        System.out.println("Tags have been added to the resource.");  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [TagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;  
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [TagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Python (Boto3) API 參考中的 [TagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

# 列出資源的標籤

使用 [ListTagsForResource](#) 動作來列出 AWS HealthImaging 中 [資料存放區](#) 和 [映像集](#) 的標籤。下列程式碼範例說明如何搭配 AWS Management Console AWS CLI、和 AWS SDKs 使用 `ListTagsForResource` 動作。如需詳細資訊，請參閱 AWS 一般參考指南中的[標記您的 AWS 資源](#)。

## 列出資源的標籤

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

1. 開啟 HealthImaging 主控台 [資料存放區頁面](#)。
2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟。

3. 選擇詳細資訊索引標籤。

在標籤區段下，會列出所有資料存放區標籤。

### AWS CLI 和 SDKs

#### CLI

##### AWS CLI

###### 範例 1：列出資料存放區的資源標籤

下列 `list-tags-for-resource` 程式碼範例會列出資料存放區的標籤。

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

輸出：

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

```
    }  
}
```

## 範例 2：列出影像集的資源標籤

下列list-tags-for-resource程式碼範例會列出影像集的標籤。

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

輸出：

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

如需詳細資訊，請參閱 [AWS HealthImaging 開發人員指南中的使用 HealthImaging 標記資源](#)。 AWS HealthImaging

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ListTagsForResource](#)。

## Java

### SDK for Java 2.x

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

```
        System.exit(1);
    }

    return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [ListTagsForResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [ListTagsForResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

    def list_tags_for_resource(self, resource_arn):
        """
        List the tags for a resource.

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

```
)  
raise  
else:  
    return tags["tags"]
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [ListTagsForResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

## 取消標記資源

使用 [UntagResource](#) 動作取消標記 AWS HealthImaging 中的 [資料存放區](#) 和 [映像集](#)。下列程式碼範例說明如何搭配 AWS Management Console AWS CLI、和 AWS SDKs 使用 UntagResource 動作。如需詳細資訊，請參閱 AWS 一般參考 指南中的 [標記您的 AWS 資源](#)。

### 取消標記資源

根據您對 AWS HealthImaging 的存取偏好設定選擇選單。

### AWS 主控台

- 開啟 HealthImaging 主控台 [資料存放區頁面](#)。

## 2. 選擇資料存放區。

資料存放區詳細資訊頁面隨即開啟。

3. 選擇詳細資訊索引標籤。
4. 在標籤區段下，選擇管理標籤。

隨即開啟管理標籤頁面。

5. 選擇您要移除的標籤旁的移除。
6. 選擇 Save changes (儲存變更)。

## AWS CLI 和 SDKs

### CLI

#### AWS CLI

##### 範例 1：取消標記資料存放區

下列untag-resource程式碼範例會取消標記資料存放區。

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

此命令不會產生輸出。

##### 範例 2：取消標記影像集

下列untag-resource程式碼範例會取消標記影像集。

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

此命令不會產生輸出。

如需詳細資訊，請參閱 [AWS HealthImaging 開發人員指南中的使用 HealthImaging 標記資源](#)。

AWS HealthImaging

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [UntagResource](#)。

## Java

### SDK for Java 2.x

```
public static void untagMedicalImagingResource(MedicalImagingClient  
medicalImagingClient,  
        String resourceArn,  
        Collection<String> tagKeys) {  
    try {  
        UntagResourceRequest untagResourceRequest =  
UntagResourceRequest.builder()  
            .resourceArn(resourceArn)  
            .tagKeys(tagKeys)  
            .build();  
  
        medicalImagingClient.untagResource(untagResourceRequest);  
  
        System.out.println("Tags have been removed from the resource.");  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [UntagResource](#)。

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;  
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [UntagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱 SDK AWS for Python (Boto3) API 參考中的 [UntagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

### 可用性範例

找不到所需的內容嗎？使用此頁面右側的提供意見回饋連結來請求程式碼範例。

# HealthImaging AWS SDKs的程式碼範例

下列程式碼範例示範如何使用 HealthImaging 搭配 AWS 軟體開發套件 (SDK)。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含入門相關資訊和舊版 SDK 的詳細資訊。

## 開始使用

### Hello HealthImaging

下列程式碼範例示範如何開始使用 HealthImaging。

#### C++

##### SDK for C++

CMakeLists.txt CMake 檔案的程式碼。

```
# 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 來源檔案的程式碼。

```
#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;
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

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

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

```
import logging  
import boto3  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
  
def hello_medical_imaging(medical_imaging_client):  
    """  
    Use the AWS SDK for Python (Boto3) to create an 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.getPaginator("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"))
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## 程式碼範例

- [HealthImaging AWS SDKs的基本範例](#)
  - [Hello HealthImaging](#)
  - [HealthImaging 使用 AWS SDKs的動作](#)
    - [CopyImageSet 搭配 AWS SDK 或 CLI 使用](#)
    - [CreateDatastore 搭配 AWS SDK 或 CLI 使用](#)
    - [DeleteDatastore 搭配 AWS SDK 或 CLI 使用](#)

- [DeleteImageSet 搭配 AWS SDK 或 CLI 使用](#)
- [GetDICOMImportJob 搭配 AWS SDK 或 CLI 使用](#)
- [GetDatastore 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageFrame 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageSet 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageSetMetadata 搭配 AWS SDK 或 CLI 使用](#)
- [ListDICOMImportJobs 搭配 AWS SDK 或 CLI 使用](#)
- [ListDatastores 搭配 AWS SDK 或 CLI 使用](#)
- [ListImageSetVersions 搭配 AWS SDK 或 CLI 使用](#)
- [ListTagsForResource 搭配 AWS SDK 或 CLI 使用](#)
- [SearchImageSets 搭配 AWS SDK 或 CLI 使用](#)
- [StartDICOMImportJob 搭配 AWS SDK 或 CLI 使用](#)
- [TagResource 搭配 AWS SDK 或 CLI 使用](#)
- [UntagResource 搭配 AWS SDK 或 CLI 使用](#)
- [UpdateImageSetMetadata 搭配 AWS SDK 或 CLI 使用](#)
- [HealthImaging using AWS SDKs 的案例](#)
  - [使用 AWS SDK 開始使用 HealthImaging 影像集和影像影格](#)
  - [使用 AWS SDK 標記 HealthImaging 資料存放區](#)
  - [使用 AWS SDK 標記 HealthImaging 映像集](#)

## HealthImaging AWS SDKs的基本範例

下列程式碼範例示範如何 AWS HealthImaging 搭配 AWS SDKs 使用 的基本概念。

### 範例

- [Hello HealthImaging](#)
- [HealthImaging 使用 AWS SDKs的動作](#)
  - [CopyImageSet 搭配 AWS SDK 或 CLI 使用](#)
  - [CreateDatastore 搭配 AWS SDK 或 CLI 使用](#)
  - [DeleteDatastore 搭配 AWS SDK 或 CLI 使用](#)
- [DeleteImageSet 搭配 AWS SDK 或 CLI 使用](#)

- [GetDICOMImportJob 搭配 AWS SDK 或 CLI 使用](#)
- [GetDatastore 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageFrame 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageSet 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageSetMetadata 搭配 AWS SDK 或 CLI 使用](#)
- [ListDICOMImportJobs 搭配 AWS SDK 或 CLI 使用](#)
- [ListDatastores 搭配 AWS SDK 或 CLI 使用](#)
- [ListImageSetVersions 搭配 AWS SDK 或 CLI 使用](#)
- [ListTagsForResource 搭配 AWS SDK 或 CLI 使用](#)
- [SearchImageSets 搭配 AWS SDK 或 CLI 使用](#)
- [StartDICOMImportJob 搭配 AWS SDK 或 CLI 使用](#)
- [TagResource 搭配 AWS SDK 或 CLI 使用](#)
- [UntagResource 搭配 AWS SDK 或 CLI 使用](#)
- [UpdateImageSetMetadata 搭配 AWS SDK 或 CLI 使用](#)

## Hello HealthImaging

下列程式碼範例示範如何開始使用 HealthImaging。

C++

SDK for C++

CMakeLists.txt CMake 檔案的程式碼。

```
# 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 來源檔案的程式碼。

```
#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;
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

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

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

logger = logging.getLogger(__name__)
```

```
def hello_medical_imaging(medical_imaging_client):
    """
    Use the AWS SDK for Python (Boto3) to create an 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.getPaginator("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"))
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## HealthImaging 使用 AWS SDKs 的動作

下列程式碼範例示範如何使用 AWS SDKs 執行個別 HealthImaging 動作。每個範例均包含 GitHub 的連結，您可以在連結中找到設定和執行程式碼的相關說明。

這些摘錄會呼叫 HealthImaging API，是必須在內容中執行之大型程式的程式碼摘錄。您可以在 [HealthImaging using AWS SDKs 的案例](#) 中查看內容中的動作。

下列範例僅包含最常使用的動作。如需完整清單，請參閱 [《AWS HealthImaging API 參考》](#)。

### 範例

- [CopyImageSet 搭配 AWS SDK 或 CLI 使用](#)
- [CreateDatastore 搭配 AWS SDK 或 CLI 使用](#)
- [DeleteDatastore 搭配 AWS SDK 或 CLI 使用](#)
- [DeleteImageSet 搭配 AWS SDK 或 CLI 使用](#)
- [GetDICOMImportJob 搭配 AWS SDK 或 CLI 使用](#)
- [GetDatastore 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageFrame 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageSet 搭配 AWS SDK 或 CLI 使用](#)
- [GetImageSetMetadata 搭配 AWS SDK 或 CLI 使用](#)
- [ListDICOMImportJobs 搭配 AWS SDK 或 CLI 使用](#)
- [ListDatastores 搭配 AWS SDK 或 CLI 使用](#)
- [ListImageSetVersions 搭配 AWS SDK 或 CLI 使用](#)
- [ListTagsForResource 搭配 AWS SDK 或 CLI 使用](#)
- [SearchImageSets 搭配 AWS SDK 或 CLI 使用](#)
- [StartDICOMImportJob 搭配 AWS SDK 或 CLI 使用](#)
- [TagResource 搭配 AWS SDK 或 CLI 使用](#)
- [UntagResource 搭配 AWS SDK 或 CLI 使用](#)
- [UpdateImageSetMetadata 搭配 AWS SDK 或 CLI 使用](#)

## CopyImageSet 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 CopyImageSet。

CLI

AWS CLI

範例 1：複製沒有目的地的影像集。

下列copy-image-set範例會複製沒有目的地的影像集。

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

輸出：

```
{
  "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"
}
```

範例 2：複製具有目的地的映像集。

下列copy-image-set範例會複製具有目的地的映像集。

```
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"} }'
```

輸出：

```
{
  "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"
}
```

範例 3：將來源映像集的執行個體子集複製到目的地映像集。

下列copy-image-set範例會將來源映像集的一個 DICOM 執行個體複製到目的地映像集。提供強制參數來覆寫病患、檢查和序列層級屬性中的不一致。

```
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
```

輸出：

```
{
  "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"
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[複製映像集](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[CopyImageSet](#)。

## Java

### SDK for Java 2.x

```
/**
 * Copy an AWS HealthImaging image set.
 *
 * @param medicalImagingClient - The AWS HealthImaging client object.
 * @param datastoreId           - The datastore ID.
 * @param imageSetId             - The image set ID.
 * @param latestVersionId       - The version ID.
```

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

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

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

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

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

        // Optionally designate a destination image set.
        if (destinationImageSetId != null) {
            copyImageSetBuilder =
copyImageSetBuilder.destinationImageSet(CopyDestinationImageSet.builder()
```

```
        .imageSetId(destinationImageSetId)
        .latestVersionId(destinationVersionId)
        .build());
    }

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

    CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

用於建立可複製屬性的公用程式函數。

```
/**
 * 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [CopyImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

用於複製映像集的公用程式函數。

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

```
/**  
 * @param {string} datastoreId - The ID of the data store.  
 * @param {string} imageSetId - The source image set ID.  
 * @param {string} sourceVersionId - The source version ID.  
 * @param {string} destinationImageSetId - The optional ID of the destination  
 * image set.  
 * @param {string} destinationVersionId - The optional version ID of the  
 * destination image set.  
 * @param {boolean} force - Force the copy action.  
 * @param {[string]} copySubsets - A subset of instance IDs to copy.  
 */  
export const copyImageSet = async (  
    datastoreId = "xxxxxxxxxxxx",  
    imageSetId = "xxxxxxxxxxxx",  
    sourceVersionId = "1",  
    destinationImageSetId = "",  
    destinationVersionId = "",  
    force = false,  
    copySubsets = [],  
) => {  
    try {  
        const params = {  
            datastoreId: datastoreId,  
            sourceImageSetId: imageSetId,  
            copyImageSetInformation: {  
                sourceImageSet: { latestVersionId: sourceVersionId },  
            },  
            force: force,  
        };  
        if (destinationImageSetId !== "" && destinationVersionId !== "") {  
            params.copyImageSetInformation.destinationImageSet = {  
                imageSetId: destinationImageSetId,  
                latestVersionId: destinationVersionId,  
            };  
        }  
  
        if (copySubsets.length > 0) {  
            let copySubsetsJson;  
            copySubsetsJson = {  
                SchemaVersion: 1.1,  
                Study: {  
                    Series: {  
                        imageSetId: {  
                            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: 'xxxxxxxxxxxxxx',
//     destinationImageSetProperties: {
//         createdAt: 2023-09-27T19:46:21.824Z,
//         imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxx',
//         imageSetId: 'xxxxxxxxxxxxxxxx',
//         imageSetState: 'LOCKED',
//         imageSetWorkflowStatus: 'COPYING',
//         latestVersionId: '1',
//         updatedAt: 2023-09-27T19:46:21.824Z
//     },
//     sourceImageSetProperties: {
//         createdAt: 2023-09-22T14:49:26.427Z,
//         imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxx',
//         imageSetId: 'xxxxxxxxxxxxxxxx',
//         imageSetState: 'LOCKED',
//         imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//     }
}
```

```
//           latestVersionId: '4',
//           updatedAt: 2023-09-27T19:46:21.824Z
//     }
//   }
//   return response;
} catch (err) {
  console.error(err);
}
};


```

複製沒有目的地的映像集。

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


```

使用目的地複製映像集。

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


```

使用目的地複製影像集的子集，並強制複製。

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

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

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [CopyImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

用於複製映像集的公用程式函數。

```
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.
        """

        response = self.health_imaging_client.copy_image_set(
            ImageSetId=image_set_id,
            DatastoreId=datastore_id,
            DestinationImageSetId=destination_image_set_id,
            DestinationVersionId=destination_version_id,
            Force=force,
            Subsets=subsets,
        )
```

```
: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"].append(
                subset
            )

    copy_image_set_information["sourceImageSet"]["DICOMCopies"] = {
        "copyableAttributes": 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"]
```

複製沒有目的地的映像集。

```
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,
)
```

使用目的地複製映像集。

```
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,
)
```

複製影像集的子集。

```
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,
)
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [CopyImageSet](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## CreateDatastore 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 CreateDatastore。

Bash

### AWS CLI 使用 Bash 指令碼

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

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

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

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

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

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

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

local error_code=${?}

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

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

echo "$response"

return 0
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 建立資料存放區

下列create-datastore程式碼範例會建立名為 的資料存放區my-datastore。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [建立資料存放區](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [CreateDatastore](#)。

## Java

### SDK for Java 2.x

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     datastoreStatus: 'CREATING'
    // }
    return response;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

```

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [CreateDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## DeleteDatastore 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 DeleteDatastore。

## Bash

### AWS CLI 使用 Bash 指令碼

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

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

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

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

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

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

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

local error_code=${?}

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

return 0
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [DeleteDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 刪除資料存放區

下列delete-datastore程式碼範例會刪除資料存放區。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[刪除資料存放區](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[DeleteDatastore](#)。

## Java

### SDK for Java 2.x

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [DeleteDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     datastoreStatus: 'DELETING'
    // }

    return response;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [DeleteDatastore](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [DeleteDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## DeleteImageSet 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 DeleteImageSet。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [影像集和影像影格入門](#)

### 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();
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [DeleteImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 刪除映像集

下列delete-image-set程式碼範例會刪除映像集。

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

輸出：

```
{  
    "imageSetWorkflowStatus": "DELETING",  
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
```

```
        "imageSetState": "LOCKED",
        "datastoreId": "12345678901234567890123456789012"
    }
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[刪除映像集](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[DeleteImageSet](#)。

## Java

### SDK for Java 2.x

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

        medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的[DeleteImageSet](#)。

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[AWS 程式碼範例儲存庫](#)中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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 = "xxxxxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxxxx",
) => {
    const response = await medicalImagingClient.send(
        new DeleteImageSetCommand({
            datastoreId,
            imageSetId,
        }),
    );
    console.log(response);
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //     datastoreId: 'xxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'DELETING'
    // }
    return response;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [DeleteImageSet](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [DeleteImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## GetDICOMImportJob 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 GetDICOMImportJob。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [影像集和影像影格入門](#)

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;
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [GetDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

若要取得 dicom 匯入任務的屬性

下列get-dicom-import-job程式碼範例會取得 dicom 匯入任務的屬性。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得匯入任務屬性](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[GetDICOMImportJob](#)。

## Java

### SDK for Java 2.x

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

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

    return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [GetDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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, 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   }
}
```

```
//           endedAt: 2023-09-19T17:29:21.753Z,
//           inputS3Uri: 's3://healthimaging-source/CTStudy/',
//           jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//           jobName: 'job_1',
//           jobStatus: 'COMPLETED',
//           outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'/',
//           submittedAt: 2023-09-19T17:27:25.143Z
//       }
// }

return response;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

        :param datastore_id: The ID of the data store.
        :param job_id: The ID of the job.
        :return: The job properties.
        """
        try:
```

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [GetDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## GetDatastore 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 GetDatastore。

Bash

### AWS CLI 使用 Bash 指令碼

```
#####
```

```
# 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
})
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [GetDatastore](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 取得資料存放區的屬性

下列get-datastore程式碼範例會取得資料存放區的屬性。

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

輸出：

```
{  
    "datastoreProperties": {  
        "datastoreId": "12345678901234567890123456789012",  
        "datastoreName": "TestDatastore123",  
        "datastoreStatus": "ACTIVE",  
        "datastoreArn": "arn:aws:medical-imaging:us-  
east-1:123456789012: datastore/1234567890123456789012",  
        "createdAt": "2022-11-15T23:33:09.643000+00:00",  
        "updatedAt": "2022-11-15T23:33:09.643000+00:00"  
    }  
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得資料存放區屬性](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[GetDatastore](#)。

## Java

### SDK for Java 2.x

```
public static DatastoreProperties  
getMedicalImageDatastore(MedicalImagingClient medicalImagingClient,
```

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

    return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [GetDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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:xxxxxxxx:datasotre/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      datastoreName: 'my_datastore',
//      datastoreStatus: 'ACTIVE',
//      updatedAt: 2023-08-04T18:50:36.239Z
//    }
//  }
// }

return response.datastoreProperties;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

```

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [GetDatastore](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## GetImageFrame 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 GetImageFrame。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [影像集和影像影格入門](#)

## 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();
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 若要取得影像集像素資料

下列get-image-frame程式碼範例會取得影像影格。

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

注意：此程式碼範例不包含輸出，因為 GetImageFrame 動作會將像素資料串流傳回至 imageframe.jph 檔案。如需解碼和檢視影像影格的資訊，請參閱 HTJ2K 解碼程式庫。

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [取得影像集像素資料](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [GetImageFrame](#)。

## Java

### SDK for Java 2.x

```
public static void getMedicalImageSetFrame(MedicalImagingClient  
medicalImagingClient,  
                                         String destinationPath,  
                                         String datastoreId,  
                                         String imagesetId,  
                                         String imageFrameId) {  
  
    try {  
        GetImageFrameRequest getImageSetMetadataRequest =  
GetImageFrameRequest.builder()  
                           .datastoreId(datastoreId)  
                           .imageSetId(imagesetId)  
  
.imageFrameInformation(ImageFrameInformation.builder()  
  
.imageFrameId(imageFrameId)  
                           .build())  
                           .build();  
  
medicalImagingClient.getImageFrame(getImageSetMetadataRequest,  
  
FileSystems.getDefault().getPath(destinationPath));  
  
System.out.println("Image frame downloaded to " +  
destinationPath);  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;
}
```

```
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

```
class MedicalImagingWrapper:  
    def __init__(self, health_imaging_client):  
        self.health_imaging_client = health_imaging_client  
  
    def get_pixel_data(  
        self, file_path_to_write, datastore_id, image_set_id, image_frame_id  
    ):  
        """  
        Get an image frame's pixel data.  
  
        :param file_path_to_write: The path to write the image frame's HTJ2K  
        encoded pixel data.  
        :param datastore_id: The ID of the data store.  
        :param image_set_id: The ID of the image set.  
        :param image_frame_id: The ID of the image frame.  
        """  
        try:  
            image_frame = self.health_imaging_client.get_image_frame(  
                datastoreId=datastore_id,  
                imageSetId=image_set_id,  
                imageFrameInformation={"imageFrameId": image_frame_id},  
            )  
            with open(file_path_to_write, "wb") as f:  
                for chunk in image_frame["imageFrameBlob"].iter_chunks():  
                    if chunk:
```

```
f.write(chunk)
except ClientError as err:
    logger.error(
        "Couldn't get image frame. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [GetImageFrame](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## GetImageSet 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 GetImageSet。

CLI

AWS CLI

若要取得影像集屬性

下列get-image-set程式碼範例會取得影像集的屬性。

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

```
--datastore-id 12345678901234567890123456789012 \
--image-set-id 18f88ac7870584f58d56256646b4d92b \
--version-id 1
```

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得映像集屬性](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[GetImageSet](#)。

## Java

### SDK for Java 2.x

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

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [GetImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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:xxxxxxxxxxxx:datasource/xxxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxx',
//     imageSetId: 'xxxxxxxxxxxxxx',
//     imageSetState: 'ACTIVE',
//     imageSetWorkflowStatus: 'CREATED',
//     updatedAt: 2023-09-22T14:49:26.427Z,
//     versionId: '1'
//   }

  return response;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetImageSet](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

    :param datastore_id: The ID of the data store.
    :param image_set_id: The ID of the image set.
    :param version_id: The optional version of the image set.
    :return: The image set properties.
    """

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [GetImageSet](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## GetImageSetMetadata 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 GetImageSetMetadata。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [影像集和影像影格入門](#)

C++

SDK for C++

取得影像集中繼資料的公用程式函數。

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

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

return outcome.IsSuccess();
}
```

取得不含 版本的映像集中繼資料。

```
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;
}
```

使用 版本取得映像集中繼資料。

```
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;
```

```
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《 API 參考》中的 [GetImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 範例 1：取得不含 版本的映像集中繼資料

下列get-image-set-metadata程式碼範例會取得映像集的中繼資料，而不指定版本。

注意：outfile 是必要的參數

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

傳回的中繼資料會以 gzip 壓縮，並存放在 studymetadata.json.gz 檔案中。若要檢視傳回 JSON 物件的內容，您必須先解壓縮它。

輸出：

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

#### 範例 2：使用 版本取得影像集中繼資料

下列get-image-set-metadata程式碼範例會取得具有指定版本之映像集的中繼資料。

注意：outfile 是必要的參數

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

傳回的中繼資料會以 gzip 壓縮，並存放在 studymetadata.json.gz 檔案中。若要檢視傳回 JSON 物件的內容，您必須先解壓縮它。

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[取得映像集中繼資料](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[GetImageSetMetadata](#)。

## Java

### SDK for Java 2.x

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

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [GetImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

取得影像集中繼資料的公用程式函數。

```
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;  
};
```

取得不含 版本的映像集中繼資料。

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

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

使用 版本取得映像集中繼資料。

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

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

取得影像集中繼資料的公用程式函數。

```
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
```

取得不含 版本的映像集中繼資料。

```
    image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
```

```
        imageSetId=image_set_id, datastoreId=datastore_id  
    )
```

使用 版本取得映像集中繼資料。

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [GetImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## ListDICOMImportJobs 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 ListDICOMImportJobs。

CLI

AWS CLI

列出 dicom 匯入任務

下列list-dicom-import-jobs程式碼範例列出 dicom 汇入任務。

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

輸出：

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

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[列出匯入任務](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[ListDICOMImportJobs](#)。

## Java

### SDK for Java 2.x

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

    try {
        ListDicomImportJobsRequest listDicomImportJobsRequest =
ListDicomImportJobsRequest.builder()
            .datastoreId(datastoreId)
            .build();
        ListDicomImportJobsResponse response =
medicalImagingClient.listDICOMImportJobs(listDicomImportJobsRequest);
        return response.jobSummaries();
```

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [ListDICOMImportJobs](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

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

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

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

    const jobSummaries = [];
    for await (const page of paginator) {
```

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

return jobSummaries;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListDICOMImportJobs](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

```
class MedicalImagingWrapper:  
    def __init__(self, health_imaging_client):  
        self.health_imaging_client = health_imaging_client  
  
    def list_dicom_import_jobs(self, datastore_id):  
        """  
        List the DICOM import jobs.  
  
        :param datastore_id: The ID of the data store.  
        :return: The list of jobs.  
        """  
        try:  
            paginator = self.health_imaging_client.getPaginator(  
                "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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ListDICOMImportJobs](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## ListDatastores 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 ListDatastores。

Bash

### AWS CLI 使用 Bash 指令碼

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

#####
# function imaging_list_datastores
#
# List the HealthImaging data stores in the account.
#
# Returns:
#     [[datastore_name, datastore_id, datastore_status]]
# And:
#     0 - If successful.
#     1 - If it fails.
#####
function imaging_list_datastores() {
    local option OPTARG # Required to use getopt command in a function.
```

```
local error_code
# bashsupport disable=BP5008
function usage() {
    echo "function imaging_list_datastores"
    echo "Lists the AWS HealthImaging data stores in the account."
    echo ""
}

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

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

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

echo "$response"

return 0
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ListDatastores](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 列出資料存放區

下列list-datastores程式碼範例列出可用的資料存放區。

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

輸出：

```
{  
    "datastoreSummaries": [  
        {  
            "datastoreId": "12345678901234567890123456789012",  
            "datastoreName": "TestDatastore123",  
            "datastoreStatus": "ACTIVE",  
            "datastoreArn": "arn:aws:medical-imaging:us-  
east-1:123456789012: datastore/1234567890123456789012",  
            "createdAt": "2022-11-15T23:33:09.643000+00:00",  
            "updatedAt": "2022-11-15T23:33:09.643000+00:00"  
        }  
    ]  
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [列出資料存放區](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ListDatastores](#)。

## Java

### SDK for Java 2.x

```
public static List<DatastoreSummary>  
listMedicalImagingDatastores(MedicalImagingClient medicalImagingClient) {
```

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

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

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

return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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:datasotre/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       datastoreName: 'my_datastore',
//       datastoreStatus: 'ACTIVE',
//       updatedAt: 2023-08-04T18:49:54.429Z
//     }
//     ...
//   ]
// }

return datastoreSummaries;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListDatastores](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

        :return: The list of data stores.
        """
        try:
            paginator =
                self.health_imaging_client.getPaginator("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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ListDatastores](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## ListImageSetVersions 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 ListImageSetVersions。

### CLI

#### AWS CLI

##### 列出映像集版本

下列list-image-set-versions程式碼範例會列出映像集的版本歷史記錄。

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

輸出：

```
{
  "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
    }
]
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[列出映像集版本](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[ListImageSetVersions](#)。

## Java

### SDK for Java 2.x

```
public static List<ImageSetProperties>
listMedicalImageSetVersions(MedicalImagingClient medicalImagingClient,
                           String datastoreId,
                           String imagesetId) {
```

```
try {
    ListImageSetVersionsRequest getImageSetRequest =
ListImageSetVersionsRequest.builder()
        .datastoreId(datastoreId)
        .imagesetId(imagesetId)
        .build();

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

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

return null;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [ListImageSetVersions](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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 it 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: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //       imageSetState: 'ACTIVE',
  //       versionId: '1'
  //     }
  //   ]
  // }
  return imageSetPropertiesList;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListImageSetVersions](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

```
class MedicalImagingWrapper:  
    def __init__(self, health_imaging_client):  
        self.health_imaging_client = health_imaging_client  
  
    def list_image_set_versions(self, datastore_id, image_set_id):  
        """  
        List the image set versions.  
  
        :param datastore_id: The ID of the data store.  
        :param image_set_id: The ID of the image set.  
        :return: The list of image set versions.  
        """  
        try:  
            paginator = self.health_imaging_client.getPaginator(  
                "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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ListImageSetVersions](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## ListTagsForResource 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 ListTagsForResource。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [標記資料存放區](#)
- [標記影像集](#)

CLI

AWS CLI

範例 1：列出資料存放區的資源標籤

下列list-tags-for-resource程式碼範例會列出資料存放區的標籤。

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

輸出：

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

#### 範例 2：列出影像集的資源標籤

下列list-tags-for-resource程式碼範例會列出影像集的標籤。

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

輸出：

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

如需詳細資訊，請參閱 [AWS HealthImaging 開發人員指南中的使用 HealthImaging 標記資源](#)。  
AWS HealthImaging

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ListTagsForResource](#)。

## Java

### SDK for Java 2.x

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [ListTagsForResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListTagsForResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

```
self.health_imaging_client = health_imaging_client

def list_tags_for_resource(self, resource_arn):
    """
    List the tags for a resource.

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ListTagsForResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## SearchImageSets 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 SearchImageSets。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [影像集和影像影格入門](#)

### C++

#### SDK for C++

用於搜尋影像集的公用程式函數。

```
///! 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 運算子。

```
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::Op
.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;  
        }  
    }  
}
```

## 使用案例 #2 : BETWEEN 運算子使用 DICOMStudyDate 和 DICOMStudyTime。

```
Aws::MedicalImaging::Model::SearchByAttributeValue useCase2StartDate;  
  
useCase2StartDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAndTime  
.WithDICOMStudyDate("19990101")  
.WithDICOMStudyTime("000000.000"));  
  
Aws::MedicalImaging::Model::SearchByAttributeValue useCase2EndDate;  
  
useCase2EndDate.SetDICOMStudyDateAndTime(Aws::MedicalImaging::Model::DICOMStudyDateAndTime  
.WithDICOMStudyDate(Aws::Utils::DateTime(std::chrono::system_clock::now()).ToLocalTimeStamp("%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 運算子。先前保留研究的時間。

```
Aws::MedicalImaging::Model::SearchByAttributeValue useCase3StartDate;
useCase3StartDate.SetCreatedAt(Aws::Utils::DateTime("20231130T00000000Z", 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;
    }
}
```

使用案例 #4：DICOMSeriesInstanceUID 上的 EQUAL 運算子和 updatedAt 上的 BETWEEN，並在 updatedAt 欄位中依 ASC 順序排序回應。

```
Aws::MedicalImaging::Model::SearchByAttributeValue useCase4StartDate;  
  
useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T00000000Z", 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;
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [SearchImageSets](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 範例 1：使用 EQUAL 運算子搜尋影像集

下列search-image-sets程式碼範例使用 EQUAL 運算子，根據特定值搜尋影像集。

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

#### search-criteria.json 的內容

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

輸出：

```
{  
    "imageSetsMetadataSummaries": [  
        {"imageSetId": "09876543210987654321098765432109",  
         "createdAt": "2022-12-06T21:40:59.429000+00:00",  
         "version": 1,  
         "DICOMTags": {  
             "DICOMStudyId": "2011201407",  
             "DICOMStudyDate": "19991122",  
             "DICOMPatientSex": "F",  
             "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",  
             "DICOMPatientBirthDate": "19201120",  
             "DICOMStudyDescription": "UNKNOWN",  
             "DICOMPatientId": "SUBJECT08701",  
             "DICOMPatientName": "Melissa844 Huel628",  
             "DICOMNumberofStudyRelatedInstances": 1,  
             "DICOMStudyTime": "140728",  
             "DICOMNumberofStudyRelatedSeries": 1  
         },  
         "updatedAt": "2022-12-06T21:40:59.429000+00:00"  
     }]  
}
```

範例 2：使用 DICOMStudyDate 和 DICOMStudyTime 以 BETWEEN 運算子搜尋影像集

下列search-image-sets程式碼範例會搜尋 1990 年 1 月 1 日至 2023 年 1 月 1 日（上午 12 : 00）之間產生 DICOM 檢查的影像集。

注意：DICOMStudyTime 是選用的。如果不存在，上午 12 : 00（一天的開始）是提供用於篩選的日期的時間值。

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

search-criteria.json 的內容

```
{  
    "filters": [{}  
    "values": [{}]
```

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

輸出：

```
{
    "imageSetsMetadataSummaries": [
        {
            "imageSetId": "09876543210987654321098765432109",
            "createdAt": "2022-12-06T21:40:59.429000+00:00",
            "version": 1,
            "DICOMTags": {
                "DICOMStudyId": "2011201407",
                "DICOMStudyDate": "19991122",
                "DICOMPatientSex": "F",
                "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",
                "DICOMPatientBirthDate": "19201120",
                "DICOMStudyDescription": "UNKNOWN",
                "DICOMPatientId": "SUBJECT08701",
                "DICOMPatientName": "Melissa844 Huel628",
                "DICOMNumberOfStudyRelatedInstances": 1,
                "DICOMStudyTime": "140728",
                "DICOMNumberOfStudyRelatedSeries": 1
            },
            "updatedAt": "2022-12-06T21:40:59.429000+00:00"
        }
    ]
}
```

範例 3：使用 createdAt 使用 BETWEEN 運算子搜尋影像集（先前保留研究的時間）

下列search-image-sets程式碼範例會搜尋 DICOM 檢查在 UTC 時區的時間範圍之間保留在 HealthImaging 中的影像集。

注意：以範例格式 ("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 的內容

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

輸出：

```
{
  "imageSetsMetadataSummaries": [
    {
      "imageSetId": "09876543210987654321098765432109",
      "createdAt": "2022-12-06T21:40:59.429000+00:00",
      "version": 1,
      "DICOMTags": {
        "DICOMStudyId": "2011201407",
        "DICOMStudyDate": "19991122",
        "DICOMPatientSex": "F",
        "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",
        "DICOMPatientBirthDate": "19201120",
        "DICOMStudyDescription": "UNKNOWN",
        "DICOMPatientId": "SUBJECT08701",
        "DICOMPatientName": "Melissa844 Huel628",
        "DICOMNumberOfStudyRelatedInstances": 1,
        "DICOMStudyTime": "140728",
        "DICOMNumberOfStudyRelatedSeries": 1
      },
      "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }
]
```

{}

範例 4：在 DICOMSeriesInstanceUID 上使用 EQUAL 運算子搜尋映像集，並在 updatedAt 上以 ASC 順序排序回應

下列search-image-sets程式碼範例會搜尋 DICOMSeriesInstanceUID 上的 EQUAL 運算子和 updatedAt 上的 BETWEEN 的影像集，並在 updatedAt 欄位中依 ASC 順序排序回應。

注意：提供範例格式的 updatedAt ("1985-04-12T23:20:50.52Z")。

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

search-criteria.json 的內容

```
{
  "filters": [
    {
      "values": [
        {
          "updatedAt": "2024-03-11T15:00:05.074000-07:00"
        },
        {
          "updatedAt": "2024-03-11T16:00:05.074000-07:00"
        }
      ],
      "operator": "BETWEEN"
    },
    {
      "values": [
        {
          "DICOMSeriesInstanceUID": "1.2.840.9999999.84710745.943275268089"
        }
      ],
      "operator": "EQUAL"
    }
  ],
  "sort": {
    "sortField": "updatedAt",
    "sortOrder": "ASC"
  }
}
```

輸出：

```
{
  "imageSetsMetadataSummaries": [
    {
      "imageSetId": "09876543210987654321098765432109",
      "createdAt": "2022-12-06T21:40:59.429000+00:00",
      "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }
  ]
}
```

```
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.9999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    ],
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的 [搜尋映像集](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [SearchImageSets](#)。

## Java

### SDK for Java 2.x

用於搜尋影像集的公用程式函數。

```
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 運算子。

```
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();
}
```

### 使用案例 #2 : BETWEEN 運算子使用 DICOMStudyDate 和 DICOMStudyTime。

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

```
.values(SearchByAttributeValue.builder())

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

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

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

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

使用案例 #3：使用 createdAt 的 BETWEEN 運算子。先前保留研究的時間。

```
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();
    }
}
```

使用案例 #4 : DICOMSeriesInstanceUID 上的 EQUAL 運算子和 updatedAt 上的 BETWEEN ,  
並在 updatedAt 欄位中依 ASC 順序排序回應。

```
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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [SearchImageSets](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

用於搜尋影像集的公用程式函數。

```
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 運算子。

```
const datastoreId = "12345678901234567890123456789012";

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

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

## 使用案例 #2 : BETWEEN 運算子使用 DICOMStudyDate 和 DICOMStudyTime。

```
const datastoreId = "12345678901234567890123456789012";

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

```
        },
    ],
};

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

使用案例 #3：使用 `createdAt` 的 `BETWEEN` 運算子。先前保留研究的時間。

```
const datastoreId = "12345678901234567890123456789012";

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

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

使用案例 #4：DICOMSeriesInstanceUID 上的 `EQUAL` 運算子和 `updatedAt` 上的 `BETWEEN`，並在 `updatedAt` 欄位中依 ASC 順序排序回應。

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [

```

```
        { updatedAt: new Date("1985-04-12T23:20:50.52Z") },
        { updatedAt: new Date() },
    ],
    operator: "BETWEEN",
},
{
    values: [
    {
        DICOMSeriesInstanceUID:
        "1.1.123.123456.1.12.1.1234567890.1234.12345678.123",
    },
],
    operator: "EQUAL",
},
],
sort: {
    sortOrder: "ASC",
    sortField: "updatedAt",
},
};

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

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [SearchImageSets](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

用於搜尋影像集的公用程式函數。

```
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":  
[{"DICOPatientId": "3524578"}]}]}  
        :return: The list of image sets.  
        """  
        try:  
            paginator =  
self.health_imaging_client.getPaginator("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 運算子。

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

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

使用案例 #2：BETWEEN 運算子使用 DICOMStudyDate 和 DICOMStudyTime。

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

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

使用案例 #3：使用 createdAt 的 BETWEEN 運算子。先前保留研究的時間。

```
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}"
)
```

使用案例 #4：DICOMSeriesInstanceUID 上的 EQUAL 運算子和 updatedAt 上的 BETWEEN，並在 updatedAt 欄位中依 ASC 順序排序回應。

```
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}")
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《AWS 適用於 Python (Boto3) 的 SDK API 參考》中的 [SearchImageSets](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## StartDICOMImportJob 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 StartDICOMImportJob。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [影像集和影像影格入門](#)

## 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();
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [StartDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## CLI

### AWS CLI

#### 啟動 dicom 匯入任務

下列start-dicom-import-job程式碼範例會啟動 dicom 匯入任務。

```
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"
```

輸出：

```
{
    "datastoreId": "12345678901234567890123456789012",
    " jobId": "09876543210987654321098765432109",
    "jobStatus": "SUBMITTED",
```

```
        "submittedAt": "2022-08-12T11:28:11.152000+00:00"  
    }  
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[啟動匯入任務](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [StartDICOMImportJob](#)。

## Java

### SDK for Java 2.x

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

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [StartDICOMImportJob](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

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

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [StartDICOMImportJob](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

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

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

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [StartDICOMImportJob](#)。

#### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## TagResource 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 TagResource。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [標記資料存放區](#)
- [標記影像集](#)

### CLI

#### AWS CLI

##### 範例 1：標記資料存放區

下列tag-resource程式碼範例會標記資料存放區。

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

此命令不會產生輸出。

##### 範例 2：標記映像集

下列tag-resource程式碼範例會標記影像集。

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

此命令不會產生輸出。

如需詳細資訊，請參閱 [AWS HealthImaging 開發人員指南中的使用 HealthImaging 標記資源](#)。  
AWS HealthImaging

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [TagResource](#)。

## Java

### SDK for Java 2.x

```
public static void tagMedicalImagingResource(MedicalImagingClient  
medicalImagingClient,  
        String resourceArn,  
        Map<String, String> tags) {  
    try {  
        TagResourceRequest tagResourceRequest = TagResourceRequest.builder()  
            .resourceArn(resourceArn)  
            .tags(tags)  
            .build();  
  
        medicalImagingClient.tagResource(tagResourceRequest);  
  
        System.out.println("Tags have been added to the resource.");  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考中的 [TagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;  
};
```

- 如需 API 詳細資訊，請參閱 適用於 JavaScript 的 AWS SDK API 參考中的 [TagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [TagResource](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## UntagResource 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 UntagResource。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [標記資料存放區](#)
- [標記影像集](#)

### CLI

#### AWS CLI

##### 範例 1：取消標記資料存放區

下列untag-resource程式碼範例會取消標記資料存放區。

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

此命令不會產生輸出。

##### 範例 2：取消標記影像集

下列untag-resource程式碼範例會取消標記影像集。

```
aws medical-imaging untag-resource \
```

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

此命令不會產生輸出。

如需詳細資訊，請參閱 [AWS HealthImaging 開發人員指南中的使用 HealthImaging 標記資源](#)。 AWS HealthImaging

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [UntagResource](#)。

## Java

### SDK for Java 2.x

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

        medicalImagingClient.untagResource(untagResourceRequest);

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

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [UntagResource](#)。

**Note**

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [UntagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [UntagResource](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## UpdateImageSetMetadata 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 UpdateImageSetMetadata。

CLI

AWS CLI

範例 1：在影像集中繼資料中插入或更新屬性

下列update-image-set-metadata範例會在影像集中繼資料中插入或更新屬性。

```
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 的內容

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

```
    }  
}
```

輸出：

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

## 範例 2：從影像集中繼資料中移除屬性

下列update-image-set-metadata範例會從影像集中繼資料中移除屬性。

```
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 的內容

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

輸出：

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

```
        "updatedAt": 1680042257.908,  
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",  
        "imageSetState": "LOCKED",  
        "createdAt": 1680027126.436,  
        "datastoreId": "12345678901234567890123456789012"  
    }  
}
```

### 範例 3：從映像集中繼資料中移除執行個體

下列update-image-set-metadata範例會從映像集中繼資料中移除執行個體。

```
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 的內容

```
{  
    "DICOMUpdates": {  
        "removableAttributes": "{\"SchemaVersion\": 1.1,\"Study\": {\"Series  
\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\":  
        {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {}}}}}}}"  
    }  
}
```

輸出：

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

### 範例 4：將映像集還原至先前的版本

下列update-image-set-metadata範例示範如何將映像集還原為舊版。CopyImageSet 和 UpdateImageSetMetadata 動作會建立新的映像集版本。

```
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"}'
```

輸出：

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

#### 範例 5：將私有 DICOM 資料元素新增至執行個體

下列update-image-set-metadata範例顯示如何將私有元素新增至映像集中的指定執行個體。DICOM 標準允許私有資料元素進行無法包含在標準資料元素中的資訊通訊。您可以使用 UpdateImageSetMetadata 動作建立、更新和刪除私有資料元素。

```
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 的內容

```
{  
    "DICOMUpdates": {  
        "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series  
\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances  
\"}}}}}}}
```

```
\": {"1.1.1.1.1.12345.123456789012.123.12345678901234.1": {"DICOM": {
    "001910F9": "97"}, "DICOMVRs": {"001910F9": "DS"}}}}"}
}
```

輸出：

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

#### 範例 6：將私有 DICOM 資料元素更新至執行個體

下列update-image-set-metadata範例顯示如何更新屬於映像集中執行個體之私有資料元素的值。

```
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 的內容

```
{
  "DICOMUpdates": {
    "updatableAttributes": "{\"SchemaVersion\": 1.1, \"Study\": {\"Series\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"Instances\": {\"1.1.1.1.1.12345.123456789012.123.12345678901234.1\": {\"DICOM\": {\"00091001\": \"GE_GENESIS_DD\"}}}}}}}"
  }
}
```

輸出：

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

### 範例 7：使用 強制參數更新 SOPInstanceUID

下列update-image-set-metadata範例示範如何使用 強制參數來覆寫 DICOM 中繼資料限制條件，以更新 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 的內容

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

輸出：

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

```
        "updatedAt": 1680042257.908,  
        "imageSetId": "53d5fdb05ca4d46ac7ca64b06545c66e",  
        "imageSetState": "LOCKED",  
        "createdAt": 1680027126.436,  
        "datastoreId": "12345678901234567890123456789012"  
    }  
}
```

如需詳細資訊，請參閱 AWS HealthImaging 開發人員指南中的[更新映像集中繼資料](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的[UpdateImageSetMetadata](#)。

## Java

### SDK for Java 2.x

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

```
.latestVersionId(versionId)
.updateImageSetMetadataUpdates(metadataUpdates)
.force(force)
.build();

        UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

### 使用案例 #1：插入或更新屬性。

```
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 屬性。

```
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：移除執行個體。

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

"1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
            "Instances": {

"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：還原至先前的版本。

```
// 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);
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x 《API 參考》中的 [UpdateImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

```
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 = "xxxxxxxxxxxx",  
    imageSetId = "xxxxxxxxxxxx",  
    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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',  
        //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',  
        //     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：插入或更新屬性並強制更新。

```
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：移除屬性。

```
// 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：移除執行個體。

```
const remove_instance = JSON.stringify({
    SchemaVersion: 1.1,
    Study: {
        Series: {
            "1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                Instances: {
                    "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：還原至舊版。

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

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

```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [UpdateImageSetMetadata](#)。

### Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

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

    def update_image_set_metadata(
            self, datastore_id, image_set_id, version_id, metadata, 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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

使用案例 #1：插入或更新屬性。

```
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 屬性。

```
# 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：移除執行個體。

```
attributes = """{  
    "SchemaVersion": 1.1,  
    "Study": {  
        "Series": {  
  
            "1.1.1.1.1.12345.123456789012.123.12345678901234.1": {  
                "Instances": {}  
  
                "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：還原至舊版。

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

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [UpdateImageSetMetadata](#)。

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## HealthImaging using AWS SDKs 的案例

下列程式碼範例示範如何在 HealthImaging AWS SDKs 中實作常見案例。這些案例說明如何透過在 HealthImaging 中呼叫多個函數或與其他函數結合，來完成特定任務 AWS 服務。每個案例均包含完整原始碼的連結，您可在連結中找到如何設定和執行程式碼的相關指示。

案例的目標是獲得中等水平的經驗，協助您了解內容中的服務動作。

### 範例

- [使用 AWS SDK 開始使用 HealthImaging 影像集和影像影格](#)
- [使用 AWS SDK 標記 HealthImaging 資料存放區](#)

- [使用 AWS SDK 標記 HealthImaging 映像集](#)

## 使用 AWS SDK 開始使用 HealthImaging 影像集和影像影格

下列程式碼範例示範如何在 HealthImaging 中匯入 DICOM 檔案和下載影像影格。

實作結構為命令列應用程式。

- 設定 DICOM 匯入的資源。
- 將 DICOM 檔案匯入資料存放區。
- 摘取匯入任務的影像集 IDs。
- 摘取影像集的影像影格 IDs。
- 下載、解碼和驗證影像影格。
- 清除資源。

C++

SDK for C++

使用必要的資源建立 AWS CloudFormation 堆疊。

```
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: ");
    }
}
```

將 DICOM 檔案複製到 Amazon S3 匯入儲存貯體。

```
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;
    }
}
```

將 DICOM 檔案匯入 Amazon S3 資料存放區。

```
bool AwsDoc::Medical_Imaging::startDicomImport(const Aws::String &dataStoreID,
```

```
const Aws::String  
&inputBucketName,  
const Aws::String &inputDirectory,  
&outputBucketName,  
const Aws::String  
&outputDirectory,  
const Aws::String &roleArn,  
const 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;
}

```

取得 DICOM 匯入任務建立的映像集。

```

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;
```

```
}
```

取得影像集的影像影格資訊。

```
bool AwsDoc::Medical_Imaging::getImageFramesForImageSet(const Aws::String
    &dataStoreID,
                                         const Aws::String
    &imageSetID,
                                         const Aws::String
    &outDirectory,
                                         Aws::Vector<ImageFrameInfo> &imageFrames,
                                         const
    Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::String fileName = outDirectory + "/" + imageSetID +
    "_metadata.json.gzip";
    bool result = false;
    if (getImageSetMetadata(dataStoreID, imageSetID, "", // Empty string for
    version ID.
                                         fileName, clientConfiguration)) {
        try {
            std::string metadataGZip;
            {
                std::ifstream inFileStream(fileName.c_str(), std::ios::binary);
                if (!inFileStream) {
                    throw std::runtime_error("Failed to open file " + fileName);
                }

                std::stringstream stringStream;
                stringStream << inFileStream.rdbuf();
                metadataGZip = stringStream.str();
            }
            std::string metadataJson = gzip::decompress(metadataGZip.data(),
                metadataGZip.size());
            // Use JMESPath to extract the image set IDs.
            // https://jmespath.org/specification.html
            jsoncons::json doc = jsoncons::json::parse(metadataJson);
            std::string jmesPathExpression = "Study.Series.*.Instances[*[]]";
            jsoncons::json instances = jsoncons::jmespath::search(doc,
            jmesPathExpression);
            for (auto &instance: instances.array_range()) {
```

```
jmesPathExpression = "DICOM.RescaleSlope";
std::string rescaleSlope = jsoncons::jmespath::search(instance,
jmesPathExpression).to_string();
jmesPathExpression = "DICOM.RescaleIntercept";
std::string rescaleIntercept =
jsoncons::jmespath::search(instance,
jmesPathExpression).to_string();

jmesPathExpression = "ImageFrames[][]";
jsoncons::json imageFramesJson =
jsoncons::jmespath::search(instance,
jmesPathExpression);

for (auto &imageFrame: imageFramesJson.array_range()) {
    ImageFrameInfo imageFrameIDs;
    imageFrameIDs.mImageSetId = imageSetID;
    imageFrameIDs.mImageFrameId = imageFrame.find(
        "ID")->value().as_string();
    imageFrameIDs.mRescaleIntercept = rescaleIntercept;
    imageFrameIDs.mRescaleSlope = rescaleSlope;
    imageFrameIDs.MinPixelValue = imageFrame.find(
        "MinPixelValue")->value().as_string();
    imageFrameIDs.MaxPixelValue = imageFrame.find(
        "MaxPixelValue")->value().as_string();

    jmesPathExpression =
"max_by(PixelDataChecksumFromBaseToFullResolution, &Width).Checksum";
    jsoncons::json checksumJson =
jsoncons::jmespath::search(imageFrame,
jmesPathExpression);
    imageFrameIDs.mFullResolutionChecksum =
checksumJson.as_integer<uint32_t>();

    imageFrames.emplace_back(imageFrameIDs);
}
}

result = true;
}
catch (const std::exception &e) {
```

```
        std::cerr << "getImageFramesForImageSet failed because " << e.what()
        << std::endl;
    }

    return result;
}

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

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

下載、解碼和驗證影像影格。

```
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.
<太后
* @param image: The OpenJPEG image struct.
* @param crc32Checksum: The CRC32 checksum.
* @return bool: Function succeeded.
*/
template<class myType>
bool verifyChecksumForImageForType(obj_image_t *image, uint32_t crc32Checksum) {
    uint32_t width = image->x1 - image->x0;
    uint32_t height = image->y1 - image->y0;
    uint32_t numChannels = image->numcomps;

    // Buffer for interleaved bitmap.
    std::vector<myType> buffer(width * height * numChannels);

    // Convert planar bitmap to interleaved bitmap.
    for (uint32_t channel = 0; channel < numChannels; 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) * numChannels + 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 += numChannels;
            }
        }
    }

    // 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;
}
```

清除資源。

```
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;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的下列主題。
  - [DeleteImageSet](#)
  - [GetDICOMImportJob](#)

- [GetImageFrame](#)
- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [StartDICOMImportJob](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

適用於 JavaScript (v3) 的 SDK

協調步驟 (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]",
});
}
```

部署資源 (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 },
);
```

```
);
```

複製 DICOM 檔案 (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.`,
);
```

開始匯入資料存放區 (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.import jobId = 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.import jobId,
    });
  };
)
```

```
    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}`,
);

```

取得影像集 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")}`,  
);
```

取得影像影格 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",
    )}\n\n`;
}
}

return output;
},
);
```

驗證影像影格 (verify-steps.js)。 [AWS HealthImaging 像素資料驗證程式庫](#)已用於驗證。

```
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>} PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */
```

```
/**  
 * @typedef {Object} Patient  
 * @property {Object} DICOM  
 */  
  
/**  
 * @typedef {}  
 * SchemaVersion: string,  
 * DatastoreID: string,  
 * ImageSetID: string,  
 * Patient: Patient,  
 * Study: Study  
 * } ImageSetMetadata  
 */  
  
/**  
 * @typedef {{ stackOutputs: {  
 *   BucketName: string,  
 *   DatastoreID: string,  
 *   RoleArn: string  
 * }, imageSetMetadata: ImageSetMetadata[] }} State  
 */  
  
export const doVerify = new ScenarioInput(  
  "doVerify",  
  "Do you want to verify the imported images?",  
  {  
    type: "confirm",  
    default: true,  
  },  
);  
  
export const decodeAndVerifyImages = new ScenarioAction(  
  "decodeAndVerifyImages",  
  async (/* @type {State} */ state) => {  
    if (!state.doVerify) {  
      process.exit(0);  
    }  
    const verificationTool = "./pixel-data-verification/index.js";  
  
    for (const metadata of state.imageSetMetadata) {  
      const datastoreId = state.stackOutputs.DatastoreId;  
      const imageSetId = metadata.ImageSetID;
```

```
for (const [seriesInstanceId, series] of Object.entries(
    metadata.Study.Series,
)) {
    for (const [sopInstanceId, _) of Object.entries(series.Instances)) {
        console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceId}
and sop ${sopInstanceId}`,
        );
        const child = spawn(
            "node",
            [
                verificationTool,
                datastoreId,
                imageSetId,
                seriesInstanceId,
                sopInstanceId,
            ],
            { stdio: "inherit" },
        );

        await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
                if (code === 0) {
                    resolve();
                } else {
                    reject(
                        new Error(
                            `Verification tool exited with code ${code} for image set
${imageSetId}`,
                        ),
                    );
                }
            });
        });
    }
},
);
```

銷毀資源 (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,
},
);
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - [DeleteImageSet](#)
  - [GetDICOMImportJob](#)

- [GetImageFrame](#)
- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [StartDICOMImportJob](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

使用必要的資源建立 AWS CloudFormation 堆疊。

```
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
```

將 DICOM 檔案複製到 Amazon S3 匯入儲存貯體。

```
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.")
```

將 DICOM 檔案匯入 Amazon S3 資料存放區。

```
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"]
```

取得 DICOM 匯入任務建立的映像集。

```
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
```

取得影像集的影像影格資訊。

```
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
```

下載、解碼和驗證影像影格。

```
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
```

清除資源。

```
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.getPaginator("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"],
        )
```

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

- 如需 API 的詳細資訊，請參閱 AWS SDK for Python (Boto3) API Reference 中的下列主題。
  - [DeleteImageSet](#)
  - [GetDICOMImportJob](#)
  - [GetImageFrame](#)
  - [GetImageSetMetadata](#)
  - [SearchImageSets](#)
  - [StartDICOMImportJob](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 AWS SDK 標記 HealthImaging 資料存放區

下列程式碼範例示範如何標記 HealthImaging 資料存放區。

Java

適用於 Java 2.x 的 SDK

標記資料存放區。

```
final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

TagResource.tagMedicalImagingResource(medicalImagingClient,
datastoreArn,
```

```
ImmutableMap.of("Deployment", "Development"));
```

標記資源的公用程式函數。

```
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);  
}  
}
```

列出資料存放區的標籤。

```
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());  
}
```

列出資源標籤的公用程式函數。

```
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;
}
```

取消標記資料存放區。

```
final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

UntagResource.untagMedicalImagingResource(medicalImagingClient,
datastoreArn,
Collections.singletonList("Deployment"));
```

用於取消標記資源的公用程式函數。

```
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);
}
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - [ListTagsForResource](#)
  - [TagResource](#)
  - [UntagResource](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

標記資料存放區。

```
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);
}
```

標記資源的公用程式函數。

```
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;
};


```

列出資料存放區的標籤。

```
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);
    }
}
```

列出資源標籤的公用程式函數。

```
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;
};


```

取消標記資料存放區。

```
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);
}
```

用於取消標記資源的公用程式函數。

```
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;
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - [ListTagsForResource](#)
  - [TagResource](#)
  - [UntagResource](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

標記資料存放區。

```
a_data_store_arn = "arn:aws:medical-imaging:us-east-1:123456789012: datastore/12345678901234567890123456789012"

medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment": "Development"})
```

標記資源的公用程式函數。

```
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
```

列出資料存放區的標籤。

```
a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

列出資源標籤的公用程式函數。

```
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"]
```

取消標記資料存放區。

```
a_data_store_arn = "arn:aws:medical-imaging:us-  
east-1:123456789012: datastore/12345678901234567890123456789012"  
  
medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

用於取消標記資源的公用程式函數。

```
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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 的詳細資訊，請參閱 AWS SDK for Python (Boto3) API Reference 中的下列主題。
  - [ListTagsForResource](#)
  - [TagResource](#)
  - [UntagResource](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 AWS SDK 標記 HealthImaging 映像集

下列程式碼範例示範如何標記 HealthImaging 映像集。

Java

適用於 Java 2.x 的 SDK

標記影像集。

```
final String imageSetArn = "arn:aws:medical-imaging:us-east-1:123456789012: datastore/12345678901234567890123456789012/imageset/12345678901234567890123456789012";

TagResource.tagMedicalImagingResource(medicalImagingClient,
imageSetArn,
ImmutableMap.of("Deployment", "Development"));
```

標記資源的公用程式函數。

```
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);  
    }  
}
```

列出影像集的標籤。

```
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());  
}
```

列出資源標籤的公用程式函數。

```
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;
}
```

取消標記影像集。

```
final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012: datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

UntagResource.untagMedicalImagingResource(medicalImagingClient,
imageSetArn,
Collections.singletonList("Deployment"));
```

用於取消標記資源的公用程式函數。

```
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);
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - [ListTagsForResource](#)
  - [TagResource](#)
  - [UntagResource](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## JavaScript

### 適用於 JavaScript (v3) 的 SDK

標記影像集。

```
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);
}
```

標記資源的公用程式函數。

```
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;
};


```

列出影像集的標籤。

```
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);
}
```

```
}
```

列出資源標籤的公用程式函數。

```
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;
};


```

取消標記影像集。

```
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);
}
```

用於取消標記資源的公用程式函數。

```
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;
};
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。
  - [ListTagsForResource](#)

- [TagResource](#)
- [UntagResource](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

## Python

### SDK for Python (Boto3)

標記影像集。

```
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"})
```

標記資源的公用程式函數。

```
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
```

列出影像集的標籤。

```
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)
```

列出資源標籤的公用程式函數。

```
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"]
```

取消標記影像集。

```
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"])
```

用於取消標記資源的公用程式函數。

```
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
```

下列程式碼會執行個體化 MedicalImagingWrapper 物件。

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

- 如需 API 的詳細資訊，請參閱 AWS SDK for Python (Boto3) API Reference 中的下列主題。
  - [ListTagsForResource](#)
  - [TagResource](#)
  - [UntagResource](#)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [將此服務與 AWS SDK 搭配使用](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

# 搭配 AWS HealthImaging 使用 DICOMweb

您可以使用 DICOMweb APIs 的表示法從 AWS HealthImaging 摳取 DICOM 物件，這是遵循醫療影像 DICOM 標準的 Web APIs。[DICOMweb](#) 此功能可讓您與使用 DICOM Part 10 二進位檔的系統互通，同時利用 HealthImaging 的雲端原生動作。本章的重點是如何使用 HealthImaging 的 DICOMweb APIs 實作來傳回 DICOMweb 回應。

## 重要

HealthImaging 會將 DICOM 資料儲存為影像集。使用 HealthImaging 雲端原生動作來管理和擷取映像集。HealthImaging 的 DICOMweb APIs 可用來傳回具有 DICOMweb 合規回應的影像集資訊。

本章中列出的 APIs 是根據 Web 型醫療影像的 [DICOMweb](#) 標準而建置。由於它們是 DICOMweb APIs 的表示法，因此不會透過 AWS CLI 和 AWS SDKs 提供。

## 主題

- [從 HealthImaging 摳取 DICOM 資料](#)
- [在 HealthImaging 中搜尋 DICOM 資料](#)

## 從 HealthImaging 摳取 DICOM 資料

AWS HealthImaging 提供 [DICOMweb WADO-RS](#) APIs 的表示法，以擷取序列和執行個體層級的資料。透過這些 APIs，您可以從 HealthImaging [資料存放區](#)擷取 DICOM 序列的所有中繼資料。您也可以擷取 DICOM 執行個體、DICOM 執行個體中繼資料和 DICOM 執行個體影格（像素資料）。HealthImaging 的 DICOMweb WADO-RS APIs 可讓您靈活地擷取存放在 HealthImaging 中的資料，並提供與舊版應用程式的互通性。

## 重要

HealthImaging 會將 DICOM 資料儲存為影像集。使用 HealthImaging 雲端原生動作來管理和擷取映像集。HealthImaging 的 DICOMweb APIs 可用來傳回具有 DICOMweb 合規回應的影像集資訊。

本節中列出的 APIs 是根據 Web 型醫療影像的 DICOMweb (WADO-RS) 標準建置。由於它們是 DICOMweb APIs 的表示法，因此不會透過 AWS CLI 和 AWS SDKs 提供。

下表說明 DICOMweb WADO-RS APIs 的所有 HealthImaging 表示法，可用於從 HealthImaging 摘取資料。

HealthImaging 表示 DICOMweb WADO-RS APIs

名稱	描述
GetDICOMSeriesMetadata	透過指定與資源相關聯的檢查和序列 UIDs，擷取 HealthImaging 資料存放區中 DICOM 序列的 DICOM 執行個體中繼資料 (.json 檔案)。請參閱 <a href="#">取得序列中繼資料</a> 。
GetDICOMInstance	透過指定與資源相關聯的序列、研究和執行個體 UIDs，從 HealthImaging 資料存放區擷取 DICOM 執行個體 (.dcm 檔案)。請參閱 <a href="#">取得執行個體</a> 。
GetDICOMInstanceMetadata	透過指定與資源相關聯的序列、研究和執行個體 UIDs，從 HealthImaging 資料存放區中的 DICOM 執行個體擷取 DICOM 執行個體中繼資料 (.json 檔案)。請參閱 <a href="#">取得執行個體中繼資料</a> 。
GetDICOMInstanceFrames	透過指定與資源相關聯的序列 UID、研究 UID、執行個體 UID 和影格編號，從 HealthImaging 資料存放區中的 DICOM 執行個體擷取單一或批次影像影格 (multipart 請求)。UIDs 請參閱 <a href="#">取得執行個體影格</a> 。

## 主題

- [從 HealthImaging 取得 DICOM 執行個體](#)
- [從 HealthImaging 取得 DICOM 執行個體中繼資料](#)
- [從 HealthImaging 取得 DICOM 執行個體影格](#)
- [從 HealthImaging 取得 DICOM 系列中繼資料](#)

## 從 HealthImaging 取得 DICOM 執行個體

使用 GetDICOMInstance動作，透過指定與資源相關聯的系列、研究和執行個體 UIDs，從 HealthImaging 資料存放區擷取 DICOM 執行個體 (.dcm 檔案)。除非提供選用的影像集參數，否則 API 只會從主要影像集傳回執行個體。您可以將 指定imageSetId為查詢參數，以擷取資料存放區中的任何執行個體（從主要或非主要影像集）。DICOM 資料可以以其儲存的傳輸語法或未壓縮 (ELE) 格式擷取。

### 取得 DICOM 執行個體 (.dcm)

1. 收集 HealthImaging datastoreId和imageSetId參數值。
2. 使用 [GetImageSetMetadata](#)動作搭配 datastoreId和 imageSetId 參數值，擷取 studyInstanceUID、seriesInstanceUID和 的相關中繼資料值sopInstanceUID。如需詳細資訊，請參閱[取得映像集中繼資料](#)。
3. 使用 datastoreId、studyInstanceUID、seriesInstanceUIDsopInstanceUID和 的值來建構請求的 URLimageSetId。若要在下列範例中檢視整個 URL 路徑，請捲動至複製按鈕。URL 的格式如下：

```
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. 準備並傳送您的請求。GetDICOMInstance使用 HTTP GET 請求搭配 [AWS Signature 第 4 版](#)簽署通訊協定。下列程式碼範例使用curl命令列工具，從 HealthImaging 取得 DICOM 執行個體 (.dcm 檔案)。

#### 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 是選用的，如果未包含，則預設為明確 VR LittleEndian。支援的傳輸語法包括：

- 明確 VR LittleEndian (ELE) - 1.2.840.10008.1.2.1 (無失真影像影格的預設值)
- 具有 RPCL 選項影像壓縮的高傳輸量 JPEG 2000 (僅限無損)  
1.2.840.10008.1.2.4.202 -- 如果執行個體存放在 HealthImaging 中  
1.2.840.10008.1.2.4.202
- JPEG 基準 (程序 1)：失真 JPEG 8 位元影像壓縮的預設傳輸語法 -  
1.2.840.10008.1.2.4.50 - 如果執行個體存放在 HealthImaging 中  
1.2.840.10008.1.2.4.50
- JPEG 2000 Image Compression 1.2.840.10008.1.2.4.91 -- 如果執行個體存放在 HealthImaging 中 1.2.840.10008.1.2.4.91
- 高輸送量 JPEG 2000 影像壓縮 - 1.2.840.10008.1.2.4.203 - 如果執行個體存放在 HealthImaging 中 1.2.840.10008.1.2.4.203
- 使用 MPEG 系列 [Transfer Syntaxes](#) (包括 MPEG2, MPEG-4 AVC/H.264 和 HEVC/H.265) 編碼的一或多個影像影格存放在 HealthImaging 中的執行個體，可以使用對應的 Transfer-syntax UID 摳取。例如，1.2.840.10008.1.2.4.100 如果執行個體儲存為 MPEG2 主要設定檔主要層級。

如需詳細資訊，請參閱[支援的傳輸語法及適用於 AWS HealthImaging 的 HTJ2K 解碼程式庫](#)。

## 從 HealthImaging 取得 DICOM 執行個體中繼資料

使用 GetDICOMInstanceMetadata動作，透過指定與資源相關聯的系列、研究和執行個體 UIDs，從 HealthImaging [資料存放](#) 區中的 DICOM 執行個體擲取中繼資料。除非提供選用的影像集參數，否則 API 只會從主要[影像集](#)傳回執行個體中繼資料。您可以透過將 [指定 imageSetId 為查詢參數](#)，擲取資料存放區中的任何執行個體中繼資料（來自主要或非主要影像集）。

### 取得 DICOM 執行個體中繼資料 (.json)

1. 收集 HealthImaging datastoreId 和 imageSetId 參數值。
2. 使用 [GetImageSetMetadata](#) 動作搭配 datastoreId 和 imageSetId 參數值，擷取 studyInstanceUID、seriesInstanceUID 和 的相關中繼資料值 sopInstanceUID。如需詳細資訊，請參閱 [取得映像集中繼資料](#)。
3. 使用 datastoreId、studyInstanceUID、seriesInstanceUIDsopInstanceUID 和 的值來建構請求的 URLimageSetId。若要在下列範例中檢視整個 URL 路徑，請捲動至複製按鈕。URL 的格式如下：

```
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. 準備並傳送您的請求。GetDICOMInstanceMetadata 使用 HTTP GET 請求搭配 [AWS Signature 第 4 版](#) 簽署通訊協定。下列程式碼範例使用 curl 命令列工具，從 HealthImaging 取得 DICOM 執行個體中繼資料 (.json 檔案)。

#### 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

中繼資料中指出的 Transfer Syntax UID 符合 HealthImaging 中的 Stored Transfer Syntax UID (StoredTransferSyntaxUID)。

## 從 HealthImaging 取得 DICOM 執行個體影格

使用 GetDICOMInstanceFrames動作，透過指定與資源相關聯的系列 UID、研究 UID、執行個體 UID 和影格編號，從 HealthImaging 資料存放 區中的 DICOM 執行個體擷取單一或批次影像影格 (multipart 請求)。UIDs 您可以透過提供映像集 ID 做為查詢參數，來指定應從中擷取執行個體影格的映像集。除非提供選用的影像集參數，否則 API 只會從主要影像集傳回執行個體影格。您可以將指定imageSetId為查詢參數，以擷取資料存放區中的任何執行個體影格（從主要或非主要影像集）。

DICOM 資料可以以其儲存的傳輸語法或未壓縮 (ELE) 格式擷取。

### 取得 DICOM 執行個體影格 (multipart)

1. 收集 HealthImaging datastoreId和imageSetId參數值。
2. 使用 [GetImageSetMetadata](#)動作搭配 datastoreId和 imageSetId 參數值來擷取 studyInstanceUID、seriesInstanceUID和 的相關中繼資料值sopInstanceUID。如需詳細資訊，請參閱[取得映像集中繼資料](#)。
3. 決定要從相關聯中繼資料擷取的影像影格，以形成 frameList 參數。frameList 參數是一個或多個非重複影格號碼的逗號分隔清單，依任何順序排列。例如，中繼資料中的第一個影像影格將是影格 1。
  - 單一影格請求：/frames/1
  - 多影格請求：/frames/1,2,3,4
4. 使用 datastoreId、studyInstanceUID、imageSetId、seriesInstanceUID sopInstanceUID和 的值來建構請求的 URLframeList。若要在下列範例中檢視整個 URL 路徑，請捲動至複製按鈕。URL 的格式如下：

```
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. 準備並傳送您的請求。GetDICOMInstanceFrames使用 HTTP GET 請求搭配 [AWS Signature 第 4 版](#)簽署通訊協定。下列程式碼範例使用curl命令列工具，從 HealthImaging 取得multipart回應中的影像影格。

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 是選用的，如果未包含，則預設為明確 VR LittleEndian。支援的傳輸語法包括：

- 明確 VR LittleEndian (ELE) - 1.2.840.10008.1.2.1 (無失真影像影格的預設值)
- 具有 RPCL 選項影像壓縮的高傳輸量 JPEG 2000 (僅限無損)  
1.2.840.10008.1.2.4.202 -- 如果執行個體存放在 HealthImaging 中  
1.2.840.10008.1.2.4.202
- JPEG 基準 (程序 1)：失真 JPEG 8 位元影像壓縮的預設傳輸語法 -  
1.2.840.10008.1.2.4.50 - 如果執行個體存放在 HealthImaging 中  
1.2.840.10008.1.2.4.50
- JPEG 2000 Image Compression 1.2.840.10008.1.2.4.91 -- 如果執行個體存放在 HealthImaging 中 1.2.840.10008.1.2.4.91
- 高輸送量 JPEG 2000 影像壓縮 - 1.2.840.10008.1.2.4.203 - 如果執行個體存放在 HealthImaging 中 1.2.840.10008.1.2.4.203
- 使用 MPEG 系列 [Transfer Syntaxes](#) (包括 MPEG2, MPEG-4 AVC/H.264 和 HEVC/H.265) 中編碼的一或多個影像影格存放在 HealthImaging 中的執行個體，可以使用對應的 Transfer-syntax UID 擷取。例如，1.2.840.10008.1.2.4.100如果執行個體儲存為 MPEG2 主要設定檔主要層級。

如需詳細資訊，請參閱[支援的傳輸語法及適用於 AWS HealthImaging 的 HTJ2K 解碼程式庫](#)。

## 從 HealthImaging 取得 DICOM 系列中繼資料

使用 GetDICOMSeriesMetadata動作從 HealthImaging 資料存放區擷取 DICOM 系列 (.json 檔案) 的中繼資料。您可以透過指定與資源相關聯的檢查和序列 UIDs，擷取 HealthImaging 資料存放區中任何主要影像集的序列中繼資料。您可以透過提供影像集 ID 做為查詢參數，擷取非主要影像集的序列中繼資料。系列中繼資料會以 DICOM JSON 格式傳回。

### 若要取得 DICOM 系列中繼資料 (.json)

1. 收集 HealthImaging datastoreId 和 imageSetId 參數值。
2. 使用 datastoreId、seriesInstanceUID、studyInstanceUID 和 選用 的值來建構請求的 URLimageSetId。若要在下列範例中檢視整個 URL 路徑，請捲動至複製按鈕。URL 的格式如下：

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

3. 準備並傳送您的請求。GetDICOMSeriesMetadata 使用具有 [AWS Signature 第 4 版簽署通訊協定的 HTTP GET](#) 請求。下列程式碼範例使用 curl 命令列工具，從 HealthImaging 取得中繼資料 (.json 檔案)。

#### 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'
```

使用選用 imageSetId 參數。

#### 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

- 需要 `imageSetId` 參數才能擷取非主要影像集的序列中繼資料。如果 `seriesInstanceUID` 指定 `datastoreId`、`、` (沒有) `studyInstanceUID`，則 `GetDICOMInstanceMetadata` 動作只會傳回主要影像集的序列中繼資料 `imagesetID`。

## 在 HealthImaging 中搜尋 DICOM 資料

AWS HealthImaging 提供 [DICOMweb QIDO-RS APIs](#) 的表示法，以依病患 ID 搜尋研究、序列和執行個體，並接收其唯一識別符以供進一步使用。HealthImaging 的 DICOMweb QIDO-RS APIs 可讓您靈活地搜尋存放在 HealthImaging 中的資料，並提供與舊版應用程式的互通性。

### 重要

HealthImaging 的 DICOMweb APIs 可用來傳回 QIDO-RS 的影像集資訊。除非另有說明，否則 HealthImaging DICOMweb APIs 只會參考 [影像集](#)。使用 HealthImaging [雲端原生動作](#)，或 DICOMweb 動作的選用影像集參數來擷取非主要影像集。HealthImaging 的 DICOMweb APIs 可用來傳回具有 DICOMweb 合規回應的影像集資訊。

HealthImaging DICOMweb QIDO-RS 動作最多可傳回 10,000 筆記錄。如果存在超過 10,000 個資源，則無法透過 QIDO-RS 動作擷取，但可以透過 DICOMweb WADO-RS 動作或 [雲端原生動作](#) 擷取。

本節中列出的 APIs 是根據 Web 型醫療影像的 DICOMweb (QIDO-RS) 標準而建置。它們不會透過 AWS CLI AWS SDKs 提供。

## HealthImaging 的 DICOMweb 搜尋 API APIs

下表說明可用於在 HealthImaging 中搜尋資料之 DICOMweb QIDO-RS APIs 的所有 HealthImaging 表示法。

HealthImaging 表示 DICOMweb QIDO-RS APIs

名稱	描述
SearchDICOMStudies	使用 GET 請求指定搜尋查詢元素，在 HealthImaging 中搜尋 DICOM 檢查。研究搜尋結果會以 JSON 格式傳回，依上次更新、日期遞減（最晚到最舊）排序。請參閱 <a href="#">搜尋研究</a> 。
SearchDICOMSeries	使用 GET 請求指定搜尋查詢元素，在 HealthImaging 中搜尋 DICOM 系列。系列搜尋結果會以 JSON 格式傳回，依遞增順序排序 Series Number (0020, 0011)（最舊到最新）。請參閱 <a href="#">搜尋系列</a> 。
SearchDICOMInstances	使用 GET 請求指定搜尋查詢元素，在 HealthImaging 中搜尋 DICOM 執行個體。執行個體搜尋結果會以 JSON 格式傳回，依遞增順序 Instance Number (0020, 0013)（最舊到最新）排序。請參閱 <a href="#">搜尋執行個體</a> 。

## HealthImaging 支援的 DICOMweb 查詢類型

HealthImaging 支援研究、系列和 SOP 執行個體層級的 QIDO-RS 階層資源查詢。使用 HealthImaging 的 QIDO-RS 階層搜尋時：

- 搜尋研究會傳回研究清單
- 搜尋檢查的序列需要已知的 StudyInstanceUID 並傳回序列清單
- 搜尋執行個體清單需要已知 StudyInstanceUID 和 SeriesInstanceUID

下表說明在 HealthImaging 中搜尋資料的支援 QIDO-RS 階層式查詢類型。

## HealthImaging 支援的 QIDO-RS 查詢類型

查詢類型	範例
屬性值查詢	<p>搜尋 檢查中的所有序列，其中 modality=CT。</p> <pre>.../studies/1.3.6.1.4.1.145 19.5.2.1.6279.6001.10137060 5276577556143013894866/series? 00080060=CT</pre> <p>搜尋病患 ID 和檢查日期分別是這些值的所有檢查。</p> <pre>.../studies?PatientID=1123581 3&amp;StudyDate=20130509</pre>
關鍵字查詢	<p>使用SeriesInstanceUID 關鍵字搜尋所有序列。</p> <pre>.../studies/1.3.6.1.4.1.145 19.5.2.1.6279.6001.10137060 5276577556143013894866/series?SeriesInstanceUID=1.3.6. 1.4.1.14519.5.2.1.6279.6001 .101370605276577556143013894868</pre>
標籤查詢	<p>使用以群組/元素形式傳遞的查詢參數來搜尋標籤。</p> <pre>{group}{element}，例如 0020000D</pre>
範圍查詢	<pre>...?Modality=CT&amp;StudyDate=A ABYYYYYY-BBCCYYYY</pre>
使用 limit 和 offset 的結果分頁	<pre>.../studies?limit=1&amp;offset=0&amp;00080020=20000101</pre>

查詢類型	範例
	<p>您可以使用限制和位移參數來分頁搜尋回應。 限制的預設值為 1000，最大值<a href="#">AWS HealthImaging 端點和配額</a>請參閱。</p> <p>最大限制 = 1000，最大位移 = 9000</p>

## 主題

- [在 HealthImaging 中搜尋 DICOM 檢查](#)
- [在 HealthImaging 中搜尋 DICOM 系列](#)
- [在 HealthImaging 中搜尋 DICOM 執行個體](#)

## 在 HealthImaging 中搜尋 DICOM 檢查

使用 SearchDICOMStudies API 在 HealthImaging [資料存放](#) 區中搜尋 DICOM 檢查。您可以透過建構包含支援 DICOM 資料元素（屬性）的 URL，在 HealthImaging 中搜尋 DICOM 檢查。研究搜尋結果會以 JSON 格式傳回，依上次更新、日期遞減（最晚到最舊）排序。

### 搜尋 DICOM 檢查

1. 收集 HealthImaging region 和 datastoreId 值。如需詳細資訊，請參閱[取得資料存放區屬性](#)。
2. 建構請求的 URL，包括所有適用的試驗元素。若要在下列範例中檢視整個 URL 路徑，請捲動至複製按鈕。URL 的格式如下：

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

### 的研究元素 **SearchDICOMStudies**

DICOM 元素標籤	DICOM 元素名稱
(0008,0020)	Study Date
(0008,0050)	Accession Number
(0008,0061)	Modalities in Study

DICOM 元素標籤	DICOM 元素名稱
(0008,0090)	Referring Physician Name
(0010,0010)	Patient Name
(0010,0020)	Patient ID
(0020,000D)	Study Instance UID
(0020,0010)	Study ID

3. 準備並傳送您的請求。SearchDICOMStudies 使用 HTTP GET 請求搭配 [AWS Signature 第 4 版](#) 簽署通訊協定。下列範例使用 curl 命令列工具來搜尋 DICOM 檢查的相關資訊。

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
```

研究搜尋結果會以 JSON 格式傳回，依上次更新、日期遞減（最晚到最舊）排序。

## 在 HealthImaging 中搜尋 DICOM 系列

使用 SearchDICOMSeries API 在 HealthImaging [資料存放](#) 區中搜尋 DICOM 系列。您可以透過建構包含支援 DICOM 資料元素（屬性）的 URL，在 HealthImaging 中搜尋 DICOM 系列。系列搜尋結果會以 JSON 格式傳回，依遞增順序排序（最舊到最新）。

### 搜尋 DICOM 系列

1. 收集 HealthImaging region 和 datastoreId 值。如需詳細資訊，請參閱[取得資料存放區屬性](#)。
2. 收集 StudyInstanceUID 值。如需詳細資訊，請參閱[取得映像集中繼資料](#)。

3. 建構請求的 URL，包括所有適用的序列元素。若要在下列範例中檢視整個 URL 路徑，請捲動至複製按鈕。URL 的格式如下：

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

### 的系列元素 **SearchDICOMSeries**

DICOM 元素標籤	DICOM 元素名稱
(0008,0060)	Modality
(0020,000E)	Series Instance UID

4. 準備並傳送您的請求。SearchDICOMSeries 使用 HTTP GET 請求搭配 [AWS Signature 第 4 版](#) 簽署通訊協定。下列範例使用 curl 命令列工具來搜尋 DICOM 系列資訊。

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
```

系列搜尋結果會以 JSON 格式傳回，依遞增順序排序 Series Number (0020,0011) ( 最舊到最新 )。

## 在 HealthImaging 中搜尋 DICOM 執行個體

使用 SearchDICOMInstances API 搜尋 HealthImaging 資料存放區中的 DICOM 執行個體。您可以透過建構包含支援 DICOM 資料元素（屬性）的 URL，在 HealthImaging 中搜尋 DICOM 執行個體。執行個體結果會以 JSON 格式傳回，依遞增排序（最舊到最新）。

### 搜尋 DICOM 執行個體

1. 收集 HealthImaging region 和 datastoreId 值。如需詳細資訊，請參閱[取得資料存放區屬性](#)。

2. 收集 StudyInstanceUID 和 SeriesInstanceUID 的值。如需詳細資訊，請參閱[取得映像集中繼資料](#)。
3. 建構請求的 URL，包括所有適用的搜尋元素。若要在下列範例中檢視整個 URL 路徑，請捲動至複製按鈕。URL 的格式如下：

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

### 的執行個體元素 SearchDICOMInstances

DICOM 元素標籤	DICOM 元素名稱
(0008,0016)	SOP Class UID
(0008,0018)	SOP Instance UID

4. 準備並傳送您的請求。SearchDICOMInstances 使用 HTTP GET 請求搭配[AWS Signature 第 4 版](#)簽署通訊協定。下列範例使用 curl 命令列工具來搜尋 DICOM 執行個體的相關資訊。

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
```

執行個體搜尋結果會以 JSON 格式傳回，依遞增順序排序 Instance Number (0020,0013) (最舊到最新)

# 監控 AWS HealthImaging

監控和記錄是維護 AWS HealthImaging 安全性、可靠性、可用性和效能的重要部分。 AWS 提供下列記錄和監控工具來監看 HealthImaging、報告錯誤，並在適當時採取自動動作：

- AWS CloudTrail 會擷取由您的帳戶或代表 AWS 您的帳戶發出的 API 呼叫和相關事件，並將日誌檔案交付至您指定的 Amazon S3 儲存貯體。您可以識別呼叫哪些使用者和帳戶 AWS、進行呼叫的來源 IP 地址，以及呼叫的時間。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》<https://docs.aws.amazon.com/awscloudtrail/latest/userguide/>。
- Amazon CloudWatch AWS 會即時監控您的 AWS 資源和您在 上執行的應用程式。您可以收集和追蹤指標、建立自訂儀板表，以及設定警示，在特定指標達到您指定的閾值時通知您或採取動作。例如，您可以讓 CloudWatch 追蹤 CPU 使用量或其他 Amazon EC2 執行個體指標，並在需要時自動啟動新的執行個體。如需詳細資訊，請參閱 [Amazon CloudWatch 使用者指南](#)。
- Amazon EventBridge 為無伺服器事件匯流排服務，可讓您輕鬆將應用程式與來自各種來源的資料互相連線。EventBridge 可從自己的應用程式、軟體即服務 (SaaS) 應用程式和 AWS 服務提供即時資料串流，然後將該資料路由到 Lambda 等目標。這可讓您監控在服務中發生的事件，並建置事件導向的架構。如需詳細資訊，請參閱 [Amazon EventBridge 使用者指南](#)。

## 主題

- [使用 AWS CloudTrail 搭配 HealthImaging](#)
- [搭配 HealthImaging 使用 Amazon CloudWatch](#)
- [搭配 HealthImaging 使用 Amazon EventBridge](#)

## 使用 AWS CloudTrail 搭配 HealthImaging

AWS HealthImaging 已與 服務整合 AWS CloudTrail，此服務可提供使用者、角色或 HealthImaging 中 AWS 服務所採取動作的記錄。CloudTrail 會將 HealthImaging 的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 HealthImaging 主控台的呼叫，以及對 HealthImaging API 操作的程式碼呼叫。如果您建立線索，您可以開啟 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 HealthImaging 的事件。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的事件歷史記錄檢視最新事件。您可以使用 CloudTrail 所收集的資訊，判斷對 HealthImaging 提出的請求、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱 [「AWS CloudTrail 使用者指南」](#)。

## 建立追蹤

當您建立帳戶 AWS 帳戶時，的 CloudTrail 會開啟。在 HealthImaging 中發生活動時，該活動會記錄在 CloudTrail 事件中，以及事件歷史記錄中的其他 AWS 服務事件。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱[使用 CloudTrail 事件歷史記錄檢視事件](#)。

### Note

若要在 中檢視 AWS HealthImaging 的 CloudTrail 事件歷史記錄 AWS Management Console，請前往查詢屬性功能表，選取事件來源，然後選擇 `medical-imaging.amazonaws.com`。

若要持續記錄 中的事件 AWS 帳戶，包括 HealthImaging 的事件，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。依預設，當您在主控台中建立追蹤時，該追蹤會套用至所有的 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案傳送到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中收集的事件資料。如需詳細資訊，請參閱下列內容：

- [建立追蹤的概觀](#)
- [CloudTrail 支援的服務和整合](#)
- [設定 CloudTrail 的 Amazon SNS 通知](#)
- [接收多個區域的 CloudTrail 日誌檔案和接收多個帳戶的 CloudTrail 日誌檔案](#)

### Note

AWS HealthImaging 支援兩種類型的 CloudTrail 事件：管理事件和資料事件。管理事件是每個 AWS 服務產生的一般事件，包括 HealthImaging。根據預設，日誌記錄會套用至已啟用它的每個 HealthImaging API 呼叫的管理事件。資料事件是計費的，通常保留給每秒高交易量 (tp) APIs，因此您可以選擇不讓 CloudTrail 日誌用於成本目的。

使用 HealthImaging，[AWS HealthImaging API 參考中列出的所有原生 API](#) 動作都會分類為管理事件，但 除外[GetImageFrame](#)。GetImageFrame 動作會加入 CloudTrail 做為資料事件，因此必須啟用。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的[記錄資料事件](#)。DICOMweb WADO-RS API 動作在 CloudTrail 中分類為資料事件，因此您必須選擇加入。如需詳細資訊，請參閱AWS CloudTrail 《 使用者指南》中的 [從 HealthImaging 摘取 DICOM 資料和記錄資料事件](#)。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 是否使用根或 AWS Identity and Access Management (IAM) 使用者登入資料提出請求。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 [CloudTrail userIdentity 元素](#)。

## 了解日誌項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌專案。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

下列範例顯示 HealthImaging 的 CloudTrail 日誌項目，示範 GetDICOMImportJob動作。

```
{  
    "eventVersion": "1.08",  
    "userIdentity": {  
        "type": "AssumedRole",  
        "principalId": "XXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",  
        "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/  
ce6d90ba-5fba-4456-a7bc-f9bc877597c3"  
        "accountId": "123456789012",  
        "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",  
        "sessionContext": {  
            "sessionIssuer": {  
                "type": "Role",  
                "principalId": "XXXXXXXXXXXXXXXXXXXX",  
                "arn": "arn:aws:iam::123456789012:role/TestAccessRole",  
                "accountId": "123456789012",  
                "userName": "TestAccessRole"  
            },  
            "webIdFederationData": {},  
            "attributes": {  
                "creationDate": "2022-10-28T15:52:42Z",  
                "mfaAuthenticated": "false"  
            }  
        }  
    },  
}
```

```
"eventTime": "2022-10-28T16:02:30Z",
"eventSource": "medical-imaging.amazonaws.com",
"eventName": "GetDICOMImportJob",
"awsRegion": "us-east-1",
"sourceIPAddress": "192.0.2.0",
"userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync http/Apache cfg/retry-mode/standard",
"requestParameters": {
    "jobId": "5d08d05d6aab2a27922d6260926077d4",
    "datastoreId": "12345678901234567890123456789012"
},
"responseElements": null,
"requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
"eventID": "26307f73-07f4-4276-b379-d362aa303b22",
"readOnly": true,
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "824333766656",
"eventCategory": "Management"
}
```

## 搭配 HealthImaging 使用 Amazon CloudWatch

您可以使用 CloudWatch 監控 AWS HealthImaging，這會收集原始資料並將其處理為可讀且近乎即時的指標。這些統計資料會保留 15 個月，因此您可以使用該歷史資訊，並更清楚地了解 Web 應用程式或服務的效能。您也可以設定留意特定閾值的警示，當滿足這些閾值時傳送通知或採取動作。如需詳細資訊，請參閱 [Amazon CloudWatch 使用者指南](#)。

 Note

系統會報告所有 HealthImaging APIs 指標。

下表列出 HealthImaging 的指標和維度。每個都會以使用者指定資料範圍的頻率計數顯示。

## 指標

指標	描述
呼叫計數	<p>對 APIs 呼叫次數。您可以針對帳戶或指定的資料存放區報告此問題。</p> <p>單位：計數</p> <p>有效統計資料：總和、計數</p> <p>維度：操作、資料存放區 ID、資料存放區類型</p>

您可以使用 AWS Management Console、AWS CLI 或 CloudWatch API 取得 HealthImaging 的指標。您可以透過其中一個 Amazon AWS 軟體開發套件 (SDKs) 或 CloudWatch API 工具來使用 CloudWatch API。HealthImaging 主控台會根據來自 CloudWatch API 的原始資料顯示圖形。

您必須擁有適當的 CloudWatch 許可，才能監控使用 CloudWatch 的 HealthImaging。如需詳細資訊，請參閱《[CloudWatch 使用者指南](#)》中的 [CloudWatch 的身分和存取管理](#)。CloudWatch

## 檢視 HealthImaging 指標

若要檢視指標 (CloudWatch 主控台)

1. 登入 AWS Management Console 並開啟 [CloudWatch 主控台](#)。
2. 選擇指標，選擇所有指標，然後選擇 AWS/醫療影像。
3. 選擇維度、選擇指標名稱，再選擇 Add to graph (新增至圖形)。
4. 選擇日期範圍的值。所選日期範圍的指標計數會顯示在圖形中。

## 使用 CloudWatch 建立警示

CloudWatch 警示會在指定的期間內監看單一指標，並執行一或多個動作：傳送通知至 Amazon Simple Notification Service (Amazon SNS) 主題或 Auto Scaling 政策。動作是根據指標在您指定的數個期間內相對於指定閾值的值。當警示變更狀態時，CloudWatch 也可以傳送 Amazon SNS 訊息給您。

CloudWatch 警示只會在狀態變更且在您指定的期間持續存在時叫用動作。如需詳細資訊，請參閱[使用 CloudWatch 警示](#)。

# 搭配 HealthImaging 使用 Amazon EventBridge

Amazon EventBridge 是一種無伺服器服務，該服務使用事件將應用程式元件連接在一起，讓您更輕鬆地建置可擴展的事件驅動型應用程式。EventBridge 的基礎是建立將事件路由到目標的規則。AWS HealthImaging 為 EventBridge 提供持久的狀態變更交付。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的什麼是 Amazon EventBridge？

## 主題

- [HealthImaging 事件傳送至 EventBridge](#)
- [HealthImaging 事件結構和範例](#)

## HealthImaging 事件傳送至 EventBridge

下表列出傳送至 EventBridge 處理的所有 HealthImaging 事件。

HealthImaging 事件類型	State
<strong>資料存放區事件</strong>	
建立資料存放區	CREATING
資料存放區建立失敗	CREATE_FAILED
已建立資料存放區	ACTIVE
刪除資料存放區	DELETING
已刪除資料存放區	DELETED
如需詳細資訊，請參閱 AWS HealthImaging API 參考中的 <a href="#">datastoreStatus</a> 。 HealthImaging	
<strong>匯入任務事件</strong>	
匯入任務已提交	SUBMITTED
匯入進行中的任務	IN_PROGRESS
匯入任務已完成	COMPLETED

HealthImaging 事件類型	State
匯入任務失敗	FAILED
如需詳細資訊，請參閱 AWS HealthImaging API 參考中的 <a href="#">jobStatus</a> 。 HealthImaging	
<b>影像集事件</b>	
影像集已建立	CREATED
映像集複製	COPYING
具有唯讀存取權的映像集複製	COPYING_WITH_READ_ONLY_ACCESS
複製的影像集	COPIED
映像集複製失敗	COPY_FAILED
映像集更新	UPDATING
映像集已更新	UPDATED
影像集更新失敗	UPDATE_FAILED
影像集刪除	DELETING
影像集已刪除	DELETED
如需詳細資訊，請參閱 AWS HealthImaging API 參考中的 <a href="#">ImageSetWorkflowStatus</a> 。 HealthImaging	

## HealthImaging 事件結構和範例

HealthImaging 事件是具有 JSON 結構的物件，也包含中繼資料詳細資訊。您可以使用中繼資料做為輸入，以重新建立事件或了解詳細資訊。所有相關聯的中繼資料欄位都會列在下表程式碼範例下的資料表中。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[事件結構參考](#)。

 Note

HealthImaging 事件結構的source屬性為 `aws.medical-imaging`。

## 資料存放區事件

### Data Store Creating

#### 狀態 - 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/  
bbc4f3ccbae4095a34170fddc19b13d"],  
    "detail": {  
        "imagingVersion": "1.0",  
        "datastoreId" : "bbc4f3ccbae4095a34170fddc19b13d",  
        "datastoreName": "test",  
        "datastoreStatus": "CREATING"  
    }  
}
```

### Data Store Creation Failed

#### 狀態 - 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/  
bbc4f3ccbae4095a34170fddc19b13d"],  
    "detail": {  
        "imagingVersion": "1.0",  
        "datastoreId" : "bbc4f3ccbae4095a34170fddc19b13d",  
        "datastoreName": "test",  
        "datastoreStatus": "CREATE_FAILED"  
    }  
}
```

}

## Data Store Created

### 狀態 - 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/  
bbc4f3ccbae4095a34170fddc19b13d"],  
    "detail": {  
        "imagingVersion": "1.0",  
        "datastoreId" : "bbc4f3ccbae4095a34170fddc19b13d",  
        "datastoreName": "test",  
        "datastoreStatus": "ACTIVE"  
    }  
}
```

## Data Store Deleting

### 狀態 - 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/  
bbc4f3ccbae4095a34170fddc19b13d"],  
    "detail": {  
        "imagingVersion": "1.0",  
        "datastoreId" : "bbc4f3ccbae4095a34170fddc19b13d",  
        "datastoreName": "test",  
        "datastoreStatus": "DELETING"  
    }  
}
```

{  
}

## Data Store Deleted

### 狀態 - **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/bbc4f3ccbae4095a34170fddc19b13d"],  
    "detail": {  
        "imagingVersion": "1.0",  
        "datastoreId" : "bbc4f3ccbae4095a34170fddc19b13d",  
        "datastoreName": "test",  
        "datastoreStatus": "DELETED"  
    }  
}

### 資料存放區事件 - 中繼資料描述

名稱	Type	描述
version	string	EventBridge 事件結構描述版本。
id	string	為每個事件產生的第 4 版 UUID。
detail-type	string	正在傳送的事件類型。
source	string	識別產生事件的服務。
account	string	資料存放區擁有者的 12 位數 AWS 帳戶 ID。

名稱	Type	描述
time	string	事件發生的時間。
region	string	識別資料存放區的 AWS 區域。
resources	陣列 ( 字串 )	包含資料存放區 ARN 的 JSON 陣列。
detail	object	包含事件相關資訊的 JSON 物件。
detail.imagingVersion	string	追蹤 HealthImaging 事件詳細資訊結構描述變更的版本 ID。
detail.datastoreId	string	與狀態變更事件相關聯的資料存放區 ID。
detail.datastoreName	string	資料存放區名稱。
detail.datastoreStatus	string	目前的資料存放區狀態。

## 匯入任務事件

### Import Job Submitted

#### 狀態 - 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/bbc4f3ccbae4095a34170fddc19b13d"],
  "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

### 狀態 - 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

### 狀態 - 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/
bbc4f3ccbae4095a34170fddc19b13d"],
"detail": {
    "imagingVersion": "1.0",
    "datastoreId" : "bbc4f3ccbae4095a34170fddc19b13d",
    "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
    "jobName": "test_only_1",
    "jobStatus": "COMPLETED",
    "inputS3Uri": "s3://healthimaging-test-bucket/input/",
    "outputS3Uri": "s3://healthimaging-test-bucket/output/"
}
}
```

## Import Job Failed

### 狀態 - 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/
bbc4f3ccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3ccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "FAILED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

## 匯入任務事件 - 中繼資料描述

名稱	Type	描述
version	string	EventBridge 事件結構描述版本。
id	string	為每個事件產生的第 4 版 UUID。
detail-type	string	正在傳送的事件類型。
source	string	識別產生事件的服務。
account	string	資料存放區擁有者的 12 位數 AWS 帳戶 ID。
time	string	事件發生的時間。
region	string	識別資料存放區的 AWS 區域。
resources	陣列 (字串)	包含資料存放區 ARN 的 JSON 陣列。
detail	object	包含事件相關資訊的 JSON 物件。
detail.imagingVersion	string	追蹤 HealthImaging 事件詳細資訊結構描述變更的版本 ID。
detail.datastoreId	string	產生狀態變更事件的資料存放區。
detail.jobId	string	與狀態變更事件相關聯的匯入任務 ID。
detail.jobName	string	匯入任務名稱。
detail.jobStatus	string	目前的任務狀態。

名稱	Type	描述
detail.inputS3Uri	string	S3 儲存貯體的輸入字首路徑，其中包含要匯入的 DICOM 檔案。
detail.outputS3Uri	string	S3 儲存貯體的輸出字首，其中會上傳 DICOM 匯入任務的結果。

## 影像集事件

### Image Set Created

#### 狀態 - 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/bbc4f3ccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
  "detail": {
    "imagingVersion": "1.0",
    "isPrimary": true,
    "imageSetVersion": "1",
    "datastoreId": "bbc4f3ccbae4095a34170fddc19b13d",
    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "ACTIVE",
    "imageSetWorkflowStatus": "CREATED"
  }
}
```

### Image Set Copying

#### 狀態 - 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"  
    }  
}
```

## Image Set Copying With Read Only Access

### 狀態 - **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

### 狀態 - 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/  
bbc4f3ccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],  
 "detail": {  
 "imagingVersion": "1.0",  
 "isPrimary": true,  
 "imageSetVersion": "1",  
 "datastoreId": "bbc4f3ccbae4095a34170fddc19b13d",  
 "imagesetId": "5b3a711878c34d40e888253319388649",  
 "imageSetState": "ACTIVE",  
 "imageSetWorkflowStatus": "COPIED"  
 }  
}  
}

## Image Set Copy Failed

### 狀態 - 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/  
bbc4f3ccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],  
 "detail": {  
 }

```
        "imagingVersion": "1.0",
        "isPrimary": true,
        "imageSetVersion": "1",
        "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
        "imagesetId": "5b3a711878c34d40e888253319388649",
        "imageSetState": "ACTIVE",
        "imageSetWorkflowStatus": "COPY_FAILED"
    }
}
```

## Image Set Updating

### 狀態 - 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

### 狀態 - 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

### 狀態 - 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

### 狀態 - **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

### 狀態 - **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",  
        "imageSetState": "DELETED",  
        "imageSetWorkflowStatus": "DELETED"  
    }  
}
```

```

    "imagesetId": "5b3a711878c34d40e888253319388649",
    "imageSetState": "DELETED",
    "imageSetWorkflowStatus": "DELETED"
}
}

```

## 影像集事件 - 中繼資料描述

名稱	Type	描述
version	string	EventBridge 事件結構描述版本。
id	string	為每個事件產生的第 4 版 UUID。
detail-type	string	正在傳送的事件類型。
source	string	識別產生事件的服務。
account	string	資料存放區擁有者的 12 位數 AWS 帳戶 ID。
time	string	事件發生的時間。
region	string	識別資料存放區的 AWS 區域。
resources	陣列 (字串)	包含影像集 ARN 的 JSON 陣列。
detail	object	包含事件相關資訊的 JSON 物件。
detail.imagingVersion	string	追蹤 HealthImaging 事件詳細資訊結構描述變更的版本 ID。
detail.isPrimary	boolean	指出匯入的資料是否已成功組織到受管階層，或是否有中繼資料衝突需要解決。

名稱	Type	描述
detail.imageSetVersion	string	匯入執行個體多次時，影像集版本會遞增。最新版本會覆寫存放在主要映像集中的任何較舊版本。
detail.datastoreId	string	產生狀態變更事件的資料存放區 ID。
detail.imagesetId	string	與狀態變更事件相關聯的影像集 ID。
detail.imageSetState	string	目前的映像集狀態。
detail.imageSetWorkflowStatus	string	目前的影像集工作流程狀態。

# 中的安全性 AWS HealthImaging

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構是為了滿足最安全敏感組織的需求而建置。

安全性是 AWS 和之間的共同責任。[共同責任模型](#)將其描述為雲端的安全性和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在中執行 AWS 服務的基礎設施 AWS 雲端。AWS 也為您提供可安全使用的服務。第三方稽核人員會定期測試和驗證我們安全的有效性，做為[AWS 合規計畫](#)的一部分。若要了解適用的合規計劃 AWS HealthImaging，請參閱[AWS 合規計畫範圍內的服務](#)。
- 雲端安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件可協助您了解如何在使用 HealthImaging 時套用共同責任模型。下列主題說明如何設定 HealthImaging 以符合您的安全和合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 HealthImaging 資源。

## 主題

- [AWS HealthImaging 中的資料保護](#)
- [AWS HealthImaging 的 Identity and Access Management](#)
- [AWS HealthImaging 的合規驗證](#)
- [AWS HealthImaging 中的基礎設施安全性](#)
- [使用 建立 AWS HealthImaging 資源 AWS CloudFormation](#)
- [AWS HealthImaging 和介面 VPC 端點 \(AWS PrivateLink\)](#)
- [的跨帳戶匯入 AWS HealthImaging](#)
- [AWS HealthImaging 中的彈性](#)

## AWS HealthImaging 中的資料保護

AWS [共同責任模型](#)適用於 AWS HealthImaging 中的資料保護。如此模型所述，AWS 負責保護執行所有的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱[資料隱私權常見問答集](#)。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 [AWS 共同的責任模型和 GDPR](#) 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶 登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《使用者指南》中的[使用 CloudTrail 追蹤](#)。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱[聯邦資訊處理標準 \(FIPS\) 140-3](#)。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 HealthImaging 或使用 AWS 服務 主控台、API AWS CLI或其他 AWS SDKs 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

## 主題

- [資料加密](#)
- [網路流量隱私權](#)

## 資料加密

使用 AWS HealthImaging，您可以為雲端中的靜態資料新增一層安全性，提供可擴展且有效率的加密功能。其中包含：

- 大多數 AWS 服務都提供靜態資料加密功能
- 靈活的金鑰管理選項，包括 AWS Key Management Service您可以選擇是否要 AWS 管理加密金鑰，還是完全控制自己的金鑰。
- AWS 擁有的 AWS KMS 加密金鑰
- 使用 Amazon SQS 的伺服器端加密 (SSE) 傳輸敏感資料的加密訊息佇列

此外， AWS 為您提供 APIs，以整合加密和資料保護與您在 AWS 環境中開發或部署的任何服務。

## 靜態加密

根據預設，HealthImaging 會使用服務擁有 AWS Key Management Service 的金鑰加密靜態客戶資料。或者，您可以設定 HealthImaging，使用您建立、擁有和管理的對稱客戶受管 AWS KMS 金鑰來加密靜態資料。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[建立對稱加密 KMS 金鑰](#)。

## 傳輸中加密

HealthImaging 使用 TLS 1.2 透過公有端點和後端服務加密傳輸中的資料。

## 金鑰管理

AWS KMS 金鑰 (KMS 金鑰) 是 中的主要資源 AWS Key Management Service。您也可以產生要在外部使用的資料金鑰 AWS KMS。

### AWS 擁有的 KMS 金鑰

HealthImaging 預設會使用這些金鑰自動加密潛在的敏感資訊，例如個人身分識別或靜態私有健康資訊 (PHI) 資料。AWS 擁有的 KMS 金鑰不會儲存在您的帳戶中。它們是擁有 AWS 和管理以在多個 AWS 帳戶中使用的 KMS 金鑰集合的一部分。AWS services 可以使用 AWS 擁有的 KMS 金鑰來保護您的資料。您無法檢視、管理、使用 AWS 擁有的 KMS 金鑰或稽核其使用方式。不過，您不需要執行任何工作或變更任何程式，即可保護加密資料的金鑰。

如果您使用 AWS 擁有的 KMS 金鑰，您不需要支付每月費用或使用費，而且這些金鑰不會計入您帳戶的 AWS KMS 配額。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[AWS 擁有的金鑰](#)。

### 客戶受管 KMS 金鑰

如果您想要完全控制 AWS KMS 生命週期和用量，HealthImaging 支援使用您建立、擁有和管理的對稱客戶受管 KMS 金鑰。您可以完全控制此層加密，因此能執行以下任務：

- 建立和維護金鑰政策、IAM 政策和授權
- 輪換金鑰密碼編譯資料
- 啟用和停用金鑰政策
- 新增標籤
- 建立金鑰別名

- 安排金鑰供刪除

您也可以使用 CloudTrail 來追蹤 HealthImaging AWS KMS 代表您傳送給的請求。AWS KMS 需支付額外費用。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[客戶受管金鑰](#)。

## 建立客戶受管金鑰

您可以使用 AWS Management Console 或 AWS KMS APIs 來建立對稱客戶受管金鑰。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[建立對稱加密 KMS 金鑰](#)。

金鑰政策會控制客戶受管金鑰的存取權限。每個客戶受管金鑰都必須只有一個金鑰政策，其中包含決定誰可以使用金鑰及其使用方式的陳述式。在建立客戶受管金鑰時，可以指定金鑰政策。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[管理客戶受管金鑰的存取](#)。

若要將客戶受管金鑰與 HealthImaging 資源搭配使用，必須在金鑰政策中允許[kms>CreateGrant](#)操作。這會將授予新增至客戶受管金鑰，以控制對指定 KMS 金鑰的存取，讓使用者存取 HealthImaging 所需的[授予权](#)。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[在中授予权 AWS KMS](#)。

若要將客戶受管 KMS 金鑰與 HealthImaging 資源搭配使用，必須在金鑰政策中允許下列 API 操作：

- kms:DescribeKey 提供驗證金鑰所需的客戶受管金鑰詳細資訊。這是所有操作的必要項目。
- kms:GenerateDataKey 提供存取權，以加密所有寫入操作的靜態資源。
- kms:Decrypt 可讓您存取加密資源的讀取或搜尋操作。
- kms:ReEncrypt\* 提供重新加密資源的存取權。

以下是政策陳述式範例，可讓使用者在 HealthImaging 中建立資料存放區，並以該金鑰加密：

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

## 使用客戶受管 KMS 金鑰所需的 IAM 許可

使用客戶受管 KMS 金鑰建立啟用 AWS KMS 加密的資料存放區時，建立 HealthImaging 資料存放區的使用者或角色需要金鑰政策和 IAM 政策的許可。

如需金鑰政策的詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[啟用 IAM 政策](#)。

建立儲存庫的 IAM 使用者、IAM 角色或 AWS 帳戶必須具有以下政策的許可，以及 AWS HealthImaging 的必要許可。

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

## HealthImaging 如何在 中使用授予 AWS KMS

HealthImaging 需要授予才能使用客戶受管 KMS 金鑰。當您建立使用客戶受管 KMS 金鑰加密的資料存放區時，HealthImaging 會透過傳送 [CreateGrant](#) 請求來代表您建立授予 AWS KMS。中的授予 AWS KMS 用於授予 HealthImaging 存取客戶帳戶中 KMS 金鑰的權限。

HealthImaging 代表您建立的授予不應被撤銷或淘汰。如果您撤銷或淘汰授予 HealthImaging 許可以使用您帳戶中的 AWS KMS 金鑰的授予，HealthImaging 無法存取此資料、加密推送到資料存放區的新映像資源，或在提取時解密這些資源。當您撤銷或淘汰 HealthImaging 的授予時，變更會立即發生。若要撤銷存取權，您應該刪除資料存放區，而不是撤銷授予。刪除資料存放區時，HealthImaging 會代表您淘汰授予。

## 監控 HealthImaging 的加密金鑰

您可以使用 CloudTrail 來追蹤 HealthImaging 在使用客戶受管 KMS 金鑰時代表您傳送給 AWS KMS 的請求。CloudTrail 日誌中的日誌項目會顯示在 `medical-imaging.amazonaws.com userAgent` 欄位中，以清楚區分 HealthImaging 提出的請求。

下列範例是 CreateGrant、Decrypt、GenerateDataKey 和的 CloudTrail 事件 DescribeKey，用於監控 HealthImaging 呼叫 AWS KMS 的操作，以存取客戶受管金鑰加密的資料。

以下說明如何使用 CreateGrant 來允許 HealthImaging 存取客戶提供的 KMS 金鑰，讓 HealthImaging 能夠使用該 KMS 金鑰來加密所有靜態客戶資料。

使用者不需要建立自己的授予。HealthImaging 會傳送 CreateGrant 請求至 AWS KMS，以代表您建立授予 AWS KMS 中的授予。AWS KMS 用於授予 HealthImaging 存取客戶帳戶中 AWS KMS 金鑰的權限。

```
        "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":
"ab4a9b919f6ca8eb2bd08ee72475658ee76cf639f721c9caaa3a148941bcd16",
    "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"
        }
    }
}
```

下列範例示範如何使用 GenerateDataKey 來確保使用者在儲存資料之前擁有加密資料的必要許可。

```
{
    "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"
}
```

下列範例顯示 HealthImaging 如何呼叫 Decrypt 操作，以使用儲存的加密資料金鑰來存取加密的資料。

```
{
    "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"
    },
},
"invokeBy": "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"
}
```

下列範例顯示 HealthImaging 如何使用 `DescribeKey` 操作來驗證 AWS KMS 客戶擁有的 AWS KMS 金鑰是否處於可用狀態，並在使用者無法運作時進行疑難排解。

```
{
    "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"
}
```

## 進一步了解

下列資源提供有關靜態資料加密的詳細資訊，位於《 AWS Key Management Service 開發人員指南》中的。

- [AWS KMS 概念](#)
- [的安全最佳實務 AWS KMS](#)

## 網路流量隱私權

HealthImaging 和內部部署應用程式之間，以及 HealthImaging 和 Amazon S3 之間都會保護流量。HealthImaging 與之間的流量預設 AWS Key Management Service 使用 HTTPS。

- AWS HealthImaging 是美國東部（維吉尼亞北部）、美國西部（奧勒岡）、歐洲（愛爾蘭）和亞太區域（雪梨）區域提供的區域服務。
- 對於 HealthImaging 和 Amazon S3 儲存貯體之間的流量，Transport Layer Security (TLS) 會加密 HealthImaging 和 Amazon S3 之間傳輸中的物件，以及 HealthImaging 和存取它的客戶應用程式之間，您應該只允許使用 Amazon S3 儲存貯體 IAM 政策[`aws:SecureTransport condition`](#)上的，透過 HTTPS (TLS) 加密連線。雖然 HealthImaging 目前使用公有端點來存取 Amazon S3 儲存貯體中的資料，但這並不表示資料周遊公有網際網路。HealthImaging 和 Amazon S3 之間的所有流量都會透過 AWS 網路路由，並使用 TLS 加密。

## AWS HealthImaging 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是 AWS 服務，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行身分驗證（登入）和授權（具有許可），以使用 HealthImaging 資源。IAM 是 AWS 服務您可以免費使用的。

### 主題

- [目標對象](#)
- [使用身分驗證](#)
- [使用政策管理存取權](#)
- [AWS HealthImaging 如何與 IAM 搭配使用](#)
- [AWS HealthImaging 的身分型政策範例](#)

- [AWS AWS HealthImaging 的 受管政策](#)
- [HealthImaging 中的跨服務混淆代理人預防](#)
- [故障診斷 AWS HealthImaging 身分和存取](#)

## 目標對象

使用方式 AWS Identity and Access Management (IAM) 會有所不同，取決於您在 HealthImaging 中所做的工作。

服務使用者 – 如果您使用 HealthImaging 服務來執行任務，管理員會為您提供所需的登入資料和許可。當您使用更多 HealthImaging 功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 HealthImaging 中的功能，請參閱 [故障診斷 AWS HealthImaging 身分和存取](#)。

服務管理員 – 如果您負責公司內的 HealthImaging 資源，您可能擁有 HealthImaging 的完整存取權。您的任務是判斷服務使用者應存取哪些 HealthImaging 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何搭配 HealthImaging 使用 IAM，請參閱 [AWS HealthImaging 如何與 IAM 搭配使用](#)。

IAM 管理員 – 如果您是 IAM 管理員，建議您了解如何撰寫政策以管理 HealthImaging 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的 HealthImaging 身分型政策範例，請參閱 [AWS HealthImaging 的身分型政策範例](#)。

## 使用身分驗證

身分驗證是您 AWS 使用身分憑證登入的方式。您必須以 AWS 帳戶根使用者身分、IAM 使用者身分或擔任 IAM 角色來驗證（登入 AWS）。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分身分登入。AWS IAM Identity Center (IAM Identity Center) 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱 AWS 登入《使用者指南》中的[如何登入您的 AWS 帳戶](#)。

如果您以 AWS 程式設計方式存取，AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的憑證以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需

使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的適用於 API 請求的 AWS Signature 第 4 版。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的 IAM 中的 AWS 多重要素驗證。

## AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取帳戶中的所有 AWS 服務和資源。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的需要根使用者憑證的任務。

## 聯合身分

最佳實務是，要求人類使用者，包括需要管理員存取權的使用者，使用聯合身分提供者 AWS 服務來使用臨時憑證來存取。

聯合身分是您企業使用者目錄、Web 身分提供者、AWS Directory Service、Identity Center 目錄或任何使用透過身分來源提供的登入資料 AWS 服務存取的使用者。當聯合身分存取時 AWS 帳戶，它們會擔任角色，而角色會提供臨時登入資料。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連接並同步到您自己的身分來源中的一組使用者 AWS 帳戶和群組，以便在所有 和應用程式中使用。如需 IAM Identity Center 的詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？

## IAM 使用者和群組

IAM 使用者是中的身分 AWS 帳戶，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證（例如密碼和存取金鑰）的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供臨時憑證。如需更多資訊，請參閱《IAM 使用者指南》中的[IAM 使用者的使用案例](#)。

## IAM 角色

[IAM 角色](#)是中具有特定許可 AWS 帳戶的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在中擔任 IAM 角色 AWS Management Console，您可以從[使用者切換至 IAM 角色（主控台）](#)。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的[擔任角色的方法](#)。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱《[IAM 使用者指南](#)》中的為第三方身分提供者（聯合）建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的[許可集](#)。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人（信任的主體）存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以將政策直接連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《[IAM 使用者指南](#)》中的[IAM 中的跨帳戶資源存取](#)。
- 跨服務存取 – 有些 AWS 服務使用其他中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
- 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫的委託人許可 AWS 服務，結合 AWS 服務請求向下游服務提出請求。只有在服務收到需要與其他 AWS 服務或資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱[《轉發存取工作階段》](#)。
- 服務角色 – 服務角色是服務擔任的[IAM 角色](#)，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《[IAM 使用者指南](#)》中的[建立角色以委派許可權給 AWS 服務](#)。

- 服務連結角色 – 服務連結角色是一種連結至 的服務角色。 AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。 IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的[使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式](#)。

## 使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件，當與身分或資源相關聯時， AWS 會定義其許可。當委託人（使用者、根使用者或角色工作階段）發出請求時，會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的[JSON 政策概觀](#)。

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 iam:GetRole 動作的政策。具有該政策的使用者可以從 AWS Management Console AWS CLI、或 API AWS 取得角色資訊。

## 身分型政策

身分型政策是可以附加到身分（例如 IAM 使用者、使用者群組或角色）的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的[透過客戶管理政策定義自訂 IAM 許可](#)。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策之間選擇，請參閱《IAM 使用者指南》中的[在受管政策和內嵌政策間選擇](#)。

## 資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

### 存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF 和 Amazon VPC 是支援 ACLs 的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的[存取控制清單 \(ACL\) 概觀](#)。

### 其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 Principal 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的[IAM 實體許可界限](#)。
- 服務控制政策 SCPs – SCPs 是 JSON 政策，可指定 中組織或組織單位 (OU) 的最大許可 AWS Organizations。AWS Organizations 是一種用於分組和集中管理您企業擁有 AWS 帳戶 之多個的服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱《AWS Organizations 使用者指南》中的[服務控制政策](#)。
- 資源控制政策 (RCP) - RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需 Organizations 和 RCPs 的詳細資訊，包括支援 RCPs AWS 服務的清單，請參閱 AWS Organizations 《使用者指南》中的[資源控制政策 \(RCPs\)](#)。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作

階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的[工作階段政策](#)。

## 多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的[政策評估邏輯](#)。

## AWS HealthImaging 如何與 IAM 搭配使用

在您使用 IAM 管理 HealthImaging 的存取權之前，請先了解哪些 IAM 功能可與 HealthImaging 搭配使用。

您可以搭配 AWS HealthImaging 使用的 IAM 功能

IAM 功能	HealthImaging 支援
<a href="#">身分型政策</a>	是
<a href="#">資源型政策</a>	否
<a href="#">政策動作</a>	是
<a href="#">政策資源</a>	是
<a href="#">政策條件索引鍵 (服務特定)</a>	是
<a href="#">ACL</a>	否
<a href="#">ABAC(政策中的標籤)</a>	部分
<a href="#">臨時憑證</a>	是
<a href="#">主體許可</a>	是
<a href="#">服務角色</a>	是
<a href="#">服務連結角色</a>	否

若要全面了解 HealthImaging 和其他 AWS 服務如何與大多數 IAM 功能搭配使用，請參閱《[AWS IAM 使用者指南](#)》中的與 IAM 搭配使用的 服務。

## HealthImaging 的身分型政策

支援身分型政策：是

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 [IAM JSON 政策元素參考](#)。

### HealthImaging 的身分型政策範例

若要檢視 HealthImaging 身分型政策的範例，請參閱 [AWS HealthImaging 的身分型政策範例](#)。

## HealthImaging 中的資源型政策

支援資源型政策：否

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中[指定主體](#)。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

如需啟用跨帳戶存取權，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，做為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當主體和資源位於不同位置時 AWS 帳戶，信任帳戶中的 IAM 管理員也必須授予主體實體（使用者或角色）存取資源的許可。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。如需詳細資訊，請參閱《IAM 使用者指南》中的 [IAM 中的快帳戶資源存取](#)。

## HealthImaging 的政策動作

支援政策動作：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 Action 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

若要查看 HealthImaging 動作清單，請參閱《服務授權參考》中的 [AWS HealthImaging 定義的動作](#)。

HealthImaging 中的政策動作在動作之前使用下列字首：

AWS

若要在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [  
    "AWS:action1",  
    "AWS:action2"  
]
```

若要檢視 HealthImaging 身分型政策的範例，請參閱 [AWS HealthImaging 的身分型政策範例](#)。

## HealthImaging 的政策資源

支援政策資源：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 [Amazon Resource Name \(ARN\)](#) 來指定資源。您可以針對支援特定資源類型的動作（稱為資源層級許可）來這麼做。

對於不支援資源層級許可的動作（例如列出操作），請使用萬用字元 (\*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

若要查看 HealthImaging 資源類型及其 ARNs，請參閱《服務授權參考》中的 [AWS HealthImaging 定義的資源類型](#)。若要了解您可以使用 ARN 的動作和資源，請參閱 [AWS HealthImaging 定義的動作](#)。

若要檢視 HealthImaging 身分型政策的範例，請參閱 [AWS HealthImaging 的身分型政策範例](#)。

## HealthImaging 的政策條件索引鍵

支援服務特定政策條件金鑰：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用 [條件運算子](#) 的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵，AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值，會使用邏輯 OR 操作 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需更多資訊，請參閱 IAM 使用者指南中的 [IAM 政策元素：變數和標籤](#)。

AWS 支援全域條件金鑰和服務特定的條件金鑰。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的 [AWS 全域條件內容索引鍵](#)。

若要查看 HealthImaging 條件索引鍵的清單，請參閱《服務授權參考》中的 [AWS HealthImaging 的條件索引鍵](#)。若要了解您可以使用條件金鑰的動作和資源，請參閱 [AWS HealthImaging 定義的動作](#)。

若要檢視 HealthImaging 身分型政策的範例，請參閱 [AWS HealthImaging 的身分型政策範例](#)。

## HealthImaging ACLs

支援 ACL：否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

## RBAC 搭配 HealthImaging

支援 RBAC

是

IAM 中使用的傳統授權模型稱為角色類型存取控制 (RBAC)。RBAC 會根據人員的任務角色 (在 AWS 外稱為角色) 來定義許可。如需詳細資訊，請參閱《IAM 使用者指南》中的將 ABAC 與傳統 RBAC 模型進行比較。

## ABAC 搭配 HealthImaging

支援 ABAC (政策中的標籤)：部分

### Warning

ABAC 不會透過 SearchImageSets API 動作強制執行。有權存取 SearchImageSets 動作的任何人都可以存取資料存放區中影像集的所有中繼資料。

### Note

影像集是資料存放區的子資源。若要使用 ABAC，映像集必須具有與資料存放區相同的標籤。如需詳細資訊，請參閱 [使用 AWS HealthImaging 標記資源](#)。

屬性型存取控制 (ABAC) 是一種授權策略，可根據屬性來定義許可。在 AWS，這些屬性稱為標籤。您可以將標籤連接到 IAM 實體（使用者或角色）和許多 AWS 資源。為實體和資源加上標籤是 ABAC 的第一步。您接著要設計 ABAC 政策，允許在主體的標籤與其嘗試存取的資源標籤相符時操作。

ABAC 在成長快速的環境中相當有幫助，並能在政策管理變得繁瑣時提供協助。

如需根據標籤控制存取，請使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的使用 ABAC 授權定義許可。如要查看含有設定 ABAC 步驟的教學課程，請參閱 IAM 使用者指南中的使用屬性型存取控制 (ABAC)。

## 搭配 HealthImaging 使用臨時登入資料

支援臨時憑證：是

當您使用臨時登入資料登入時，有些 AWS 服務無法運作。如需詳細資訊，包括哪些 AWS 服務使用臨時登入資料，請參閱《[AWS 服務 IAM 使用者指南](#)》中的 使用 IAM。

如果您 AWS Management Console 使用使用者名稱和密碼以外的任何方法登入，則會使用臨時登入資料。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取時，該程序會自動建立臨時登入資料。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱《IAM 使用者指南》中的[從使用者切換至 IAM 角色 \(主控台\)](#)。

您可以使用 AWS CLI 或 AWS API 手動建立臨時登入資料。然後，您可以使用這些臨時登入資料來存取 AWS。AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱[IAM 中的暫時性安全憑證](#)。

## HealthImaging 的跨服務主體許可

支援轉寄存取工作階段 (FAS)：是

當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為委託人。政策能將許可授予主體。當您使用某些服務時，您可能會執行一個動作，然後在不同的服務中觸發另一個動作。在此情況下，您必須具有執行這兩個動作的許可。若要查看動作是否需要政策中的其他相依動作，請參閱《服務授權參考》中的[AWS HealthImaging 的動作、資源和條件索引鍵](#)。

## HealthImaging 的服務角色

支援服務角色：是

服務角色是服務擔任的[IAM 角色](#)，可代您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[建立角色以委派許可權給 AWS 服務](#)。

### Warning

變更服務角色的許可可能會中斷 HealthImaging 功能。只有在 HealthImaging 提供指引時，才能編輯服務角色。

## HealthImaging 的服務連結角色

支援服務連結角色：否

服務連結角色是連結至的一種服務角色 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的中 AWS 帳戶，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服務連結角色的詳細資訊，請參閱[可搭配 IAM 運作的 AWS 服務](#)。在表格中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

## AWS HealthImaging 的身分型政策範例

根據預設，使用者和角色沒有建立或修改 HealthImaging 資源的許可。他們也無法使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 AWS API 來執行任務。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

如需了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的[建立 IAM 政策 \(主控台\)](#)。

如需 Awesome 定義的動作和資源類型的詳細資訊，包括每種資源類型的 ARNs 格式，請參閱《服務授權參考》中的[Awesome AWS 的動作、資源和條件金鑰](#)。

### 主題

- [政策最佳實務](#)
- [使用 HealthImaging 主控台](#)
- [允許使用者檢視他們自己的許可](#)

### 政策最佳實務

身分型政策會判斷您帳戶中的某人是否可以建立、存取或刪除 HealthImaging 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱 IAM 使用者指南中的[AWS 受管政策或任務職能的AWS 受管政策](#)。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的[IAM 中的政策和許可](#)。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 例如 使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南中的[IAM JSON 政策元素：條件](#)。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access

Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的[使用 IAM Access Analyzer 驗證政策](#)。

- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》[https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_credentials\\_mfa\\_configure-api-require.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html)中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的[IAM 安全最佳實務](#)。

## 使用 HealthImaging 主控台

若要存取 AWS HealthImaging 主控台，您必須擁有一組最低許可。這些許可必須允許您列出和檢視 中 HealthImaging 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為了確保使用者和角色仍然可以使用 HealthImaging 主控台，也請將 HealthImaging *ConsoleAccess*或*ReadOnly* AWS 受管政策連接到實體。如需詳細資訊，請參閱《IAM 使用者指南》中的[新增許可到使用者](#)。

## 允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ViewOwnUserInfo",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetUserPolicy",  
                "iam>ListGroupsForUser",  
                "iam>ListAttachedUserPolicies",  
                "iam>ListUserPolicies",  
                "iam GetUser"  
            ],  
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]  
        }  
    ]  
}
```

```
},
{
    "Sid": "NavigateInConsole",
    "Effect": "Allow",
    "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam>ListAttachedGroupPolicies",
        "iam>ListGroupPolicies",
        "iam>ListPolicyVersions",
        "iam>ListPolicies",
        "iam>ListUsers"
    ],
    "Resource": "*"
}
]
}
```

## AWS AWS HealthImaging 的 受管政策

AWS 受管政策是由 AWS AWS 受管政策建立和管理的獨立政策旨在為許多常見使用案例提供許可，以便您可以開始將許可指派給使用者、群組和角色。

請記住，AWS 受管政策可能不會授予特定使用案例的最低權限許可，因為這些許可可供所有 AWS 客戶使用。我們建議您定義使用案例專屬的客戶管理政策，以便進一步減少許可。

您無法變更 AWS 受管政策中定義的許可。如果 AWS 更新 受管政策中 AWS 定義的許可，則更新會影響政策連接的所有主體身分（使用者、群組和角色）。當新的 AWS 服務 啟動或新的 API 操作可用於現有 服務時，AWS 最有可能更新 AWS 受管政策。

如需詳細資訊，請參閱《IAM 使用者指南》中的 [AWS 受管政策](#)。

### 主題

- [AWS 受管政策 : AWSHealthImagingFullAccess](#)
- [AWS 受管政策 : AWSHealthImagingReadOnlyAccess](#)

- [HealthImaging AWS 受管政策的更新](#)

## AWS 受管政策 : AWSHealthImagingFullAccess

您可將 AWSHealthImagingFullAccess 政策連接到 IAM 身分。

此政策會將管理許可授予所有 HealthImaging 動作。

```
{  
    "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 受管政策 : AWSHealthImagingReadOnlyAccess

您可將 AWSHealthImagingReadOnlyAccess 政策連接到 IAM 身分。

此政策授予特定 AWS HealthImaging 動作的唯讀許可。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "medical-imaging:ListJobs",  
            "Resource": "*"  
        }  
    ]  
}
```

```
"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 受管政策的更新

檢視自此服務開始追蹤這些變更以來HealthImaging AWS 受管政策更新的詳細資訊。如需此頁面變更的自動提醒，請訂閱[發行頁面上的 RSS 摘要](#)。

變更	描述	日期
HealthImaging 已開始追蹤變更	HealthImaging 開始追蹤其 AWS 受管政策的變更。	2023 年 7 月 19 日

## HealthImaging 中的跨服務混淆代理人預防

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在 AWS 中，跨服務模擬可能會導致混淆代理人問題。在某個服務（呼叫服務）呼叫另一個服務（被呼叫服務）時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了防止這種情況，AWS 提供的工具可協助您保護所有 服務的資料，其服務主體已獲得您帳戶中資源的存取權。

我們建議您在 IAM ImportJobDataAccessRole 角色信任關係政策中使用 [aws:SourceArn](#) 和 [aws:SourceAccount](#) 全域條件內容金鑰，以限制 AWS HealthImaging 為您的資源提供其他服務的許

可。使用 `aws:SourceArn`，僅將一個資源與跨服務存取權相關聯。使用 `aws:SourceAccount`，讓該帳戶中的任何資源都與跨服務使用相關聯。如果您同時使用兩個全域條件內容索引鍵，則在相同政策陳述式中使用 `aws:SourceAccount` 值和 `aws:SourceArn` 值中參考的帳戶時，必須使用相同的帳戶 ID。

的值 `aws:SourceArn` 必須是受影響資料存放區的 ARN。如果您不知道資料存放區的完整 ARN，或如果您指定多個資料存放區，請使用 `aws:SourceArn` 全域內容條件索引鍵搭配 \* 萬用字元表示 ARN 的未知部分。例如，您可以將 `aws:SourceArn` 設定為 `arn:aws:medical-imaging:us-west-2:111122223333:datastore/*`。

在下列信任政策範例中，我們使用 `aws:SourceArn` 和 `aws:SourceAccount` 條件金鑰，根據資料存放區的 ARN 限制對服務主體的存取，以防止混淆代理人問題。

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

## 故障診斷 AWS HealthImaging 身分和存取

使用以下資訊來協助您診斷和修正使用 HealthImaging 和 IAM 時可能遇到的常見問題。

### 主題

- [我無權在 HealthImaging 中執行動作](#)
- [我未獲得執行 iam:PassRole 的授權](#)

- [我想要允許以外的人員 AWS 帳戶存取我的 HealthImaging 資源](#)

## 我無權在 HealthImaging 中執行動作

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 AWS:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
AWS:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 AWS:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

## 我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 iam:PassRole 動作，您的政策必須更新，以允許您將角色傳遞給 HealthImaging。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 的 IAM marymajor 使用者嘗試使用主控台在 HealthImaging 中執行動作時，會發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:  
iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

## 我想要允許以外的人員 AWS 帳戶存取我的 HealthImaging 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 HealthImaging 是否支援這些功能，請參閱 [AWS HealthImaging 如何與 IAM 搭配使用](#)。
- 若要了解如何提供您擁有 AWS 帳戶的資源存取權，請參閱《[IAM 使用者指南](#)》中的在您擁有 AWS 帳戶的另一個中為 IAM 使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的[將存取權提供給第三方 AWS 帳戶擁有](#)。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的[將存取權提供給外部進行身分驗證的使用者\(聯合身分\)](#)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的[IAM 中的跨帳戶資源存取](#)。

## AWS HealthImaging 的合規驗證

第三方稽核人員會在多個合規計畫中評估 AWS HealthImaging 的安全性和 AWS 合規性。對於 HealthImaging，這包括 HIPAA。

如需特定合規計劃範圍內 AWS 的服務清單，請參閱[合規計劃範圍內的 AWS 服務](#)。如需一般資訊，請參閱[AWS 合規計劃](#)。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱[在 中下載報告 AWS Artifact](#)。

您在使用 AWS HealthImaging 時的合規責任取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源以協助合規：

- [AWS Partner Solutions – Security and Compliance](#) 的自動化參考部署指南討論架構考量，並提供部署安全和以合規為中心基準環境的步驟 AWS。
- [HIPAA 安全與合規架構白皮書](#) – 此白皮書說明公司如何使用 AWS 來建立符合 HIPAA 規範的應用程式。
- [GxP Systems on AWS](#) – 此白皮書提供如何處理 AWS GxP 相關合規和安全性的資訊，並提供在 GxP 環境中使用 AWS 服務的指引。
- [AWS 合規資源](#) – 此工作手冊和指南的集合可能適用於您的產業和位置。
- [使用規則評估資源](#) – AWS Config 評估資源組態符合內部實務、產業準則和法規的程度。
- [AWS Security Hub](#) – AWS 此服務提供 內安全狀態的完整檢視 AWS，可協助您檢查是否符合安全產業標準和最佳實務。

## AWS HealthImaging 中的基礎設施安全性

作為受管服務，AWS HealthImaging 受到 [Amazon Web Services：安全程序概觀白皮書](#) 中所述 AWS 的全球網路安全程序保護。

您可以使用 AWS 已發佈的 API 呼叫，透過網路存取 HealthImaging。用戶端必須支援 Transport Layer Security (TLS) 1.3 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如 臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。您也可以使用 [AWS Security Token Service](#)(AWS STS) 產生臨時安全登入資料來簽署請求。

## 使用 建立 AWS HealthImaging 資源 AWS CloudFormation

AWS HealthImaging 已與整合 AWS CloudFormation，此服務可協助您建立和設定 AWS 資源的模型，以便減少建立和管理資源和基礎設施的時間。您可以建立範本，描述您想要的所有 AWS 資源，並為您 AWS CloudFormation 佈建和設定這些資源。

使用 時 AWS CloudFormation，您可以重複使用範本，以一致且重複地設定 HealthImaging 資源。描述您的資源一次，然後在多個 AWS 帳戶 和 區域中逐一佈建相同的資源。

### HealthImaging 和 AWS CloudFormation 範本

若要佈建和設定 HealthImaging 和相關服務的資源，您必須了解 [AWS CloudFormation 範本](#)。範本是以 JSON 或 YAML 格式化的文本檔案。這些範本說明您想要在 AWS CloudFormation 堆疊中佈建的資源。如果您不熟悉 JSON 或 YAML，您可以使用 AWS CloudFormation 設計工具來協助您開始使用 AWS CloudFormation 範本。如需更多詳細資訊，請參閱 AWS CloudFormation 使用者指南 中的 [什麼是 AWS CloudFormation 設計器？](#)。

AWS HealthImaging 支援使用 建立[資料存放區](#) AWS CloudFormation。如需詳細資訊，包括用於佈建 HealthImaging 資料存放區的 JSON 和 YAML 範本範例，請參閱AWS CloudFormation 《使用者指南》中的 [AWS HealthImaging 資源類型參考](#)。

### 進一步了解 AWS CloudFormation

若要進一步了解 AWS CloudFormation，請參閱下列資源：

- [AWS CloudFormation](#)

- [AWS CloudFormation 使用者指南](#)
- [AWS CloudFormation API 參考](#)
- [AWS CloudFormation 命令列界面使用者指南](#)

## AWS HealthImaging 和介面 VPC 端點 (AWS PrivateLink)

您可以在 VPC 和 之間建立私有連線， AWS HealthImaging 方法是建立界面 VPC 端點。界面端點採用 [AWS PrivateLink](#)，這項技術可讓您在沒有網際網路閘道、NAT 裝置、VPN 連線或 AWS Direct Connect 連線的情況下，私下存取 HealthImaging APIs。VPC 中的執行個體不需要公有 IP 地址，即可與 HealthImaging APIs 通訊。VPC 和 HealthImaging 之間的流量不會離開 Amazon 網路。

每個界面端點都是由您子網路中的一或多個[彈性網路介面](#)表示。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的界面 [VPC 端點 \(AWS PrivateLink\)](#)。

### 主題

- [HealthImaging VPC 端點的考量事項](#)
- [為 HealthImaging 建立介面 VPC 端點](#)
- [為 HealthImaging 建立 VPC 端點政策](#)

## HealthImaging VPC 端點的考量事項

在您設定 HealthImaging 的介面 VPC 端點之前，請務必檢閱 Amazon VPC 使用者指南中的[介面端點屬性和限制](#)。

HealthImaging 支援從 VPC 呼叫所有 AWS HealthImaging 動作。

## 為 HealthImaging 建立介面 VPC 端點

您可以使用 Amazon VPC 主控台或 AWS Command Line Interface () 為 HealthImaging 服務建立 VPC 端點 AWS CLI。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[建立介面端點](#)。

使用下列服務名稱為 HealthImaging 建立 VPC 端點：

- com.amazonaws.*region*.Medical-imaging
- com.amazonaws.*region*.runtime-medical-imaging
- com.amazonaws.*region*.dicom-medical-imaging

**Note**

必須啟用私有 DNS 才能使用 PrivateLink。

您可以使用區域的預設 DNS 名稱向 HealthImaging 提出 API 請求，例如 `medical-imaging.us-east-1.amazonaws.com`。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[透過介面端點存取服務](#)。

## 為 HealthImaging 建立 VPC 端點政策

您可以將端點政策連接至控制 HealthImaging 存取的 VPC 端點。此政策會指定下列資訊：

- 可執行動作的委託人
- 可執行的動作
- 可在其中執行動作的資源

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[使用 VPC 端點控制對服務的存取](#)。

### 範例：HealthImaging 動作的 VPC 端點政策

以下是 HealthImaging 端點政策的範例。連接到端點時，此政策會授予所有資源上所有主體對 HealthImaging 動作的存取權。

### 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": "*"}]}]"
```

## 的跨帳戶匯入 AWS HealthImaging

透過跨帳戶/跨區域匯入，您可以從位於其他[支援區域的](#) Amazon S3 儲存貯體將資料匯入 HealthImaging 資料存放區。您可以從位於開放資料登錄檔的 AWS 帳戶、其他[AWS 組織](#)擁有的帳戶，以及來自諸如[影像資料共用 \(IDC\)](#) 等開放資料來源匯入資料。[AWS](#)

HealthImaging 跨帳戶/跨區域匯入使用案例包括：

- 從客戶帳戶匯入 DICOM 資料的醫療影像 SaaS 產品
- 大型組織從許多 Amazon S3 輸入儲存貯體填入一個 HealthImaging 資料存放區
- 研究人員安全地跨多機構臨床試驗共用資料

### 使用跨帳戶匯入

1. Amazon S3 輸入（來源）儲存貯體擁有者必須授予 HealthImaging 資料存放區擁有者 `s3>ListBucket` 和 `s3:GetObject` 許可。
2. HealthImaging 資料存放區擁有者必須將 Amazon S3 儲存貯體新增至其 IAM `ImportJobDataAccessRole`。請參閱 [建立用於匯入的 IAM 角色](#)。
3. HealthImaging 資料存放區擁有者必須在開始匯入任務時提供 `inputOwnerAccountId` Amazon S3 輸入儲存貯體的。

#### Note

透過提供 `inputOwnerAccountId`，資料存放區擁有者會驗證輸入 Amazon S3 儲存貯體屬於指定的帳戶，以維持符合產業標準並降低潛在的安全風險。

下列startDICOMImportJob程式碼範例包含選用inputOwnerId參數，可套用至AWS CLI [啟動匯入任務](#)區段中的所有和SDK程式碼範例。

## Java

```
public static String startDicomImportJob(MedicalImagingClient  
medicalImagingClient,  
    String jobName,  
    String datastoreId,  
    String dataAccessRoleArn,  
    String inputS3Uri,  
    String outputS3Uri,  
    String inputOwnerId) {  
  
    try {  
        StartDicomImportJobRequest startDicomImportJobRequest =  
StartDicomImportJobRequest.builder()  
            .jobName(jobName)  
            .datastoreId(datastoreId)  
            .dataAccessRoleArn(dataAccessRoleArn)  
            .inputS3Uri(inputS3Uri)  
            .outputS3Uri(outputS3Uri)  
            .inputOwnerId(inputOwnerId)  
            .build();  
        StartDicomImportJobResponse response =  
medicalImagingClient.startDICOMImportJob(startDicomImportJobRequest);  
        return response.jobId();  
    } catch (MedicalImagingException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
  
    return "";  
}
```

## AWS HealthImaging 中的彈性

AWS 全球基礎設施是以 AWS 區域 和 可用區域為基礎建置。AWS 區域 提供多個實體隔離和隔離的可用區域，這些區域與低延遲、高輸送量和高度備援聯網連接。透過可用區域，您可以設計與操作的應

用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如果您需要在更大的地理距離內複製資料或應用程式，請使用 AWS 本地區域。AWS 本機區域是單一資料中心，旨在補充現有的 AWS 區域。如同所有 AWS 區域，AWS 本機區域與其他區域完全隔離 AWS 區域。

如需 AWS 區域 和 可用區域的詳細資訊，請參閱[AWS 全球基礎設施](#)。

# AWS HealthImaging 參考資料

## Note

所有原生 HealthImaging API 動作和資料類型都會在 [AWS HealthImaging API 參考](#) 中說明。

## 主題

- [DICOM 支援 AWS HealthImaging](#)
- [AWS HealthImaging 參考](#)

## DICOM 支援 AWS HealthImaging

AWS HealthImaging 支援特定 DICOM 元素和傳輸語法。熟悉支援的病患、檢查和序列層級 DICOM 資料元素，因為 HealthImaging 中繼資料金鑰是以它們為基礎。開始匯入之前，請確認您的醫學影像資料符合 HealthImaging 支援的傳輸語法和 DICOM 元素限制條件。

## Note

AWS HealthImaging 目前不支援二進位分段影像或圖示影像序列像素資料。

## 主題

- [支援的 SOP 類別](#)
- [中繼資料標準化](#)
- [支援的傳輸語法](#)
- [DICOM 元素限制條件](#)
- [DICOM 中繼資料限制條件](#)

## 支援的 SOP 類別

使用 AWS HealthImaging，您可以匯入以任何 SOP 類別 UID 編碼的 DICOM P10 Service-Object 對 (SOP) 執行個體，包括 [淘汰](#) 和 [私有](#)。所有私有屬性也會保留。

## 中繼資料標準化

當您將 DICOM P10 資料匯入 AWS HealthImaging 時，資料會轉換為包含中繼資料和影像影格的影像集（像素資料）。在轉換過程中，HealthImaging 中繼資料金鑰會根據特定版本的 DICOM 標準產生。HealthImaging 目前會根據 [DICOM PS3.6 2022b 資料字典](#)產生和支援中繼資料金鑰。

AWS HealthImaging 在病患、檢查和序列層級支援下列 DICOM 資料元素。

### 病患層級元素

#### Note

如需每個病患層級元素的詳細說明，請參閱 [DICOM 資料元素的登錄檔](#)。

AWS HealthImaging 支援下列病患層級元素：

#### **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

## 研究層級元素

### Note

如需每個研究層級元素的詳細說明，請參閱 [DICOM 資料元素的登錄檔](#)。

AWS HealthImaging 支援下列研究層級元素：

### 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

## 系列層級元素

### Note

如需每個序列層級元素的詳細說明，請參閱 [DICOM 資料元素的登錄檔](#)。

AWS HealthImaging 支援下列序列層級元素：

### 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

## 支援的傳輸語法

AWS HealthImaging 支援匯入具有不同傳輸語法的 DICOM P10 檔案。有些檔案會在匯入期間保留其原始傳輸語法編碼，而其他檔案預設會轉碼為 HTJ2K 無失真。下列範例顯示 HealthImaging 如何在傳回的 中繼資料 中 StoredTransferSyntaxUID 記錄每個執行個體的 [GetImageSetMetadata](#)。

"Instances": {

```
"999.999.2.19941105.134500.2.101": {
    "StoredTransferSyntaxUID": "1.2.840.10008.1.2.4.50",
    "ImageFrames": [{ ...
```

### Note

檢視下表時，請記住這些要點：

- 以星號 (\*) 標記的傳輸語法 UID 項目表示檔案在匯入期間以原始編碼格式儲存。對於這些檔案，StoredTransferSyntaxUID位於執行個體中繼資料中的 符合原始傳輸語法。
- 沒有星號標記的傳輸語法 UID 項目表示檔案在匯入期間轉碼為 HTJ2K 無失真，並存放在 HealthImaging 中。對於這些檔案，StoredTransferSyntaxUID位於執行個體中繼資料中的 是具有 RPCL 選項影像壓縮的高輸送量 JPEG 2000 - 僅限無損 (1.2.840.10008.1.2.4.202)。
- 如果StoredTransferSyntaxUID金鑰不存在或設定為 null，您可以假設其編碼為 HTJ2K 無失真 RPCL (1.2.840.10008.1.2.4.202)。

## HealthImaging 支援的傳輸語法

傳輸語法 UID	傳輸語法名稱
1.2.840.10008.1.2	隱含 VR Endian : DICOM 的預設傳輸語法
1.2.840.10008.1.2.1* (二進位分段會保留原始編碼，而非二進位分段會轉碼為 HTJ2K 無失真 RPCL)	明確 VR LittleEndian
1.2.840.10008.1.2.1.99	Deflated Explicit VR LittleEndian
1.2.840.10008.1.2.2	明確 VR BigEndian
1.2.840.10008.1.2.4.50*	JPEG 基準 (程序 1)：失真 JPEG 8 位元影像壓縮的預設傳輸語法
1.2.840.10008.1.2.4.51	JPEG 基準 (程序 2 和 4)：失真 JPEG 12 位元影像壓縮的預設傳輸語法 (僅限程序 4)
1.2.840.10008.1.2.4.57	JPEG 無失真非階層式 (程序 14)

傳輸語法 UID	傳輸語法名稱
1.2.840.10008.1.2.4.70	JPEG 無失真、非階層式、一階預測（程序 14【選擇值 1】）：無失真 JPEG 影像壓縮的預設傳輸語法
1.2.840.10008.1.2.4.80	JPEG-LS 無失真影像壓縮
1.2.840.10008.1.2.4.81	JPEG-LS 失真（近乎失真）影像壓縮
1.2.840.10008.1.2.4.90	JPEG 2000 影像壓縮（僅限無損）
1.2.840.10008.1.2.4.91*	JPEG 2000 影像壓縮
1.2.840.10008.1.2.4.201	高輸送量 JPEG 2000 影像壓縮（僅無損）
1.2.840.10008.1.2.4.202	具有 RPCL 選項影像壓縮的高輸送量 JPEG 2000（僅無損）
1.2.840.10008.1.2.4.203*	高輸送量 JPEG 2000 影像壓縮
1.2.840.10008.1.2.5	RLE 無失真
1.2.840.10008.1.2.4.100*、1.2.840.1008.1.2.4.100.1*	MPEG2 主要設定檔主要層級
1.2.840.10008.1.2.4.101*、1.2.840.1008.1.2.4.101.1*	高階 MPEG2 主要設定檔
1.2.840.10008.1.2.4.102*、1.2.840.1008.1.2.4.102.1*	MPEG-4 AVC/H.264 高設定檔/4.1 級
1.2.840.10008.1.2.4.103*、1.2.840.1008.1.2.4.103.1*	MPEG-4 AVC/H.264 BD 相容 High Profile/Level 4.1
1.2.840.10008.1.2.4.104*、1.2.840.1008.1.2.4.104.1*	適用於 2D 影片的 MPEG-4 AVC/H.264 高設定檔/4.2 級
1.2.840.10008.1.2.4.105*、1.2.840.1008.1.2.4.105.1*	MPEG-4 AVC/H.264 高設定檔 / ITU-T H.264 影片的 4.2 級

傳輸語法 UID	傳輸語法名稱
1.2.840.10008.1.2.4.106*、1.2.840.1008.1.2.4.106.1*	MPEG-4 AVC/H.264 立體聲高設定檔 / ITU-T H.264 影片的 4.2 級
1.2.840.10008.1.2.4.107*	HEVC/H.265 主要設定檔/5.1 級影片
1.2.840.10008.1.2.4.108*	HEVC/H.265 主要 10 設定檔/層級 5.1

\*在匯入期間保留原始傳輸語法編碼

## DICOM 元素限制條件

將醫療影像資料匯入 AWS HealthImaging 時，最大長度限制會套用至下列 DICOM 元素。若要成功匯入，請確定您的資料未超過長度上限限制。

### 匯入期間的 DICOM 元素限制條件

HealthImaging 關鍵字	DICOM 關鍵字	DICOM 金鑰	長度限制
DICOMPatientName	病患名稱	(0010 , 0010)	最小值：0，最大值：256
DICOMPatientId	病患 ID	(0010 , 0020)	最小值：0，最大值：256
DICOMPatientBirthDate	PatientBirthDate	(0010 , 0030)	最小值：0，最大值：18
DICOMPatientSex	PatientSex	(0010 , 0040)	最小值：0，最大值：16
DICOMStudyInstanceUID	StudyInstanceUID	(0020 , 000D)	最小值：0，最大值：256
DICOMStudyId	StudyID	(0020 , 0010)	最小值：0，最大值：16
DICOMStudyDescription	StudyDescription	(0008 , 1030)	最小值：0，最大值：64

HealthImaging 關鍵字	DICOM 關鍵字	DICOM 金鑰	長度限制
DICOMNumberOfStudyRelatedSeries	與試驗相關的系列數量	(0020, 1206)	最小值：0，最大值：1,000,000
DICOMNumberOfStudyRelatedInstances	NumberOfStudyRelated 執行個體	(0020, 1208)	最小值：0，最大值：10,000
DICOMAccessionNumber	AccessionNumber	(0008, 0050)	最小值：0，最大值：256
DICOMStudyDate	StudyDate	(0008, 0020)	最小值：0，最大值：18
DICOMStudyTime	StudyTime	(0008, 0030)	最小值：0，最大值：28

## DICOM 中繼資料限制條件

當您使用 `UpdateImageSetMetadata` 更新 HealthImaging [中繼資料](#)屬性時，會套用下列 DICOM 限制條件。

- 無法更新或移除Patient/Study/Series/執行個體層級屬性上的私有屬性，除非更新限制條件同時適用於 `updateableAttributes` 和 `removableAttributes`
- 無法更新下列 AWS HealthImaging 產生的屬性：`SchemaVersion`、`DatastoreId`、`ImageSetID`、`PixelData`、`Checksum`、`Width`、`Height`、`MinFrameSizeInBytes`
- 除非已設定 `force` 旗標，否則無法更新下列 DICOM 屬性：`Tag.PixelData`、`Tag.StudyInstanceUID`、`Tag.SeriesInstanceUID`、`Tag.SOPInstanceUID`、`Tag.StudyID`
- 除非已設定 `force` 旗標，否則無法使用 VR 類型 SQ ( 巢狀屬性 ) 更新屬性
- 除非已設定 `force` 旗標，否則無法更新多值屬性
- 除非已設定 `force` 旗標，否則無法使用與屬性 VR 類型不相容的值更新屬性
- 除非已設定 `force` 旗標，否則無法根據 DICOM 標準更新不被視為有效屬性的屬性

- 無法跨模組更新屬性。例如，如果在客戶承載請求中的研究層級提供病患層級屬性，則請求可能會失效。
- 如果關聯的屬性模組不存在於現有 中，則無法更新屬性ImageSetMetadata。例如，seriesInstanceUID如果具有的系列seriesInstanceUID不存在於現有的映像集中繼資料中，則不允許您更新 的屬性。

## AWS HealthImaging 參考

本節包含與 AWS HealthImaging 相關的支援資料。

### 主題

- [AWS HealthImaging 端點和配額](#)
- [AWS HealthImaging 限流限制](#)
- [AWS HealthImaging 像素資料驗證](#)
- [適用於 AWS HealthImaging 的 HTJ2K 解碼程式庫](#)
- [AWS HealthImaging 範例專案](#)
- [將此服務與 AWS SDK 搭配使用](#)

## AWS HealthImaging 端點和配額

下列主題包含 AWS HealthImaging 服務端點和配額的相關資訊。

### 主題

- [服務端點](#)
- [Service Quotas](#)

## 服務端點

服務端點是 URL，可識別主機和連接埠做為 Web 服務的進入點。每個 Web 服務請求都會包含一個端點。大多數 AWS 服務為特定區域提供端點，以實現更快的連線。下表列出 AWS HealthImaging 的服務端點。

區域名稱	區域	端點	通訊協定
美國東部 (維吉尼亞 北部)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS
美國西部 (奧勒岡)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS
亞太區域 (悉尼)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS
歐洲 (愛 爾蘭)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS

如果您使用 HTTP 請求來呼叫 AWS HealthImaging 動作，您必須根據呼叫的動作使用不同的端點。下列功能表列出 HTTP 請求的可用服務端點及其支援的動作。

### HTTP 請求支援的 API 動作

data store, import, tagging

下列資料存放區、匯入和標記動作可透過端點存取：

[https://medical-imaging.\*region\*.amazonaws.com](https://medical-imaging.<i>region</i>.amazonaws.com)

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- StartDICOMImportJob

- GetDICOMImportJob
- ListDICOMImportJobs
- TagResource
- ListTagsForResource
- UntagResource

## image set

下列影像集動作可透過端點存取：

`https://runtime-medical-imaging.region.amazonaws.com`

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet
- DeleteImageSet

## DICOMweb

HealthImaging 提供 DICOMweb Retrieve WADO-RS 服務的表示法。如需詳細資訊，請參閱[從 HealthImaging 擷取 DICOM 資料](#)。

下列 DICOMweb 服務可透過端點存取：

`https://dicom-medical-imaging.region.amazonaws.com`

- GetDICOMInstance
- GetDICOMInstanceMetadata
- GetDICOMInstanceFrames

## Service Quotas

服務配額定義為您 AWS 帳戶中資源、動作和項目的最大值。

### Note

對於可調整配額，您可以使用[Service Quotas 主控台](#)請求增加配額。如需詳細資訊，請參閱「Service Quotas 使用者指南」中的[請求提高配額](#)。

下表列出 AWS HealthImaging 的預設配額。

名稱	預設	可調整	描述
每個資料存放區的並行 CopyImageSet 請求上限	每個受支援的區域：100	是	目前 AWS 區域中每個資料存放區的最大並行 CopyImageSet 請求數
每個資料存放區的並行 DeleteImageSet 請求上限	每個受支援的區域：100	是	目前 AWS 區域中每個資料存放區的最大並行 DeleteImageSet 請求數

名稱	預設	可調整	描述
每個資料存放區的並行 UpdateImageSetMetadata 請求上限	每個受支援的區域 : 100	<a href="#">是</a>	目前 AWS 區域中每個資料存放區的最大並行 UpdateImageSetMetadata 請求數
每個資料存放區的並行匯入任務上限	ap-southeast-2 : 20 每個其他支援的區域 : 100	<a href="#">是</a>	目前 AWS 區域中每個資料存放區的並行匯入任務數量上限
資料存放區上限	每個受支援的區域 : 10	<a href="#">是</a>	目前 AWS 區域中作用中資料存放區的數量上限
每個 CopyImageSet 請求允許複製的 ImageFrames 數量上限	每個受支援的區域 : 1,000	<a href="#">是</a>	目前 AWS 區域中每個 CopyImageSet 請求允許複製 ImageFrames 數量上限
DICOM 傳輸任務中的檔案數目上限	每個受支援的區域 : 5,000	<a href="#">是</a>	目前 AWS 區域中 DICOM 傳輸任務中的檔案數目上限
DICOM 傳輸任務中的巢狀資料夾數量上限	每個受支援的區域 : 10,000	否	目前 AWS 區域中 DICOM 傳輸任務中的巢狀資料夾數量上限
UpdateImageSetMetadata 接受的最大承載大小限制 (KB)	每個支援的區域 : 10 KB	<a href="#">是</a>	UpdateImageSetMetadata 在目前 AWS 區域中接受的最大承載大小限制 (KB)
DICOM 傳輸任務中所有檔案的大小上限 (以 GB 為單位)	每個支援的區域 : 10 GB	否	目前 AWS 區域中 DICOM 傳輸任務中所有檔案的大小上限 (以 GB 為單位)

名稱	預設	可調整	描述
DICOM 匯入任務中每個 DICOM P10 檔案的大小上限 (以 GB 為單位)	每個支援的區域 : 4 GB	否	目前 AWS 區域中 DICOM 匯入任務中每個 DICOM P10 檔案的大小上限 (以 GB 為單位)
每個匯入、複製和 UpdateImageSet 的 ImageSetMetadata 大小上限 (以 MB 為單位)	每個支援的區域 : 50 MB	是	目前 AWS 區域中每個匯入、複製和 UpdateImageSet 的 ImageSetMetadata 大小上限 (以 MB 為單位)

## AWS HealthImaging 限流限制

AWS 您的帳戶具有適用於 AWS HealthImaging API 動作的限流限制。對於所有動作，如果超過限流限制，則會擲回 ThrottlingException 錯誤。如需詳細資訊，請參閱 [AWS HealthImaging API 參考](#)。

### Note

調節限制可針對所有 HealthImaging API 動作進行調整。若要請求調節限制調整，請聯絡 [AWS 支援中心](#)。若要建立案例，請登入 AWS 您的帳戶，然後選擇建立案例。

下表列出原生 HealthImaging 動作和 [DICOMweb 服務的表示方式](#)的限流限制。

### AWS HealthImaging 限流限制

動作	調節速率	調節爆量
CreateDatastore	0.085 tps	1 tps
GetDatastore	10 tps	20 tps
ListDatastores	5 tps	10 tps
DeleteDatastore	0.085 tps	1 tps

動作	調節速率	調節爆量
StartDICOMImportJob	0.25 tps	1 tps
GetDICOMImportJob	25 tps	50 tps
ListDICOMImportJobs	10 tps	20 tps
SearchImageSets	25 tps	50 tps
GetImageSet	25 tps	50 tps
GetImageSetMetadata	50 tps	100 tps
GetImageFrame	1000 tps	2000 tps
ListImageSetVersions	25 tps	50 tps
UpdateImageSetMetadata	0.25 tps	1 tps
CopyImageSet	0.25 tps	1 tps
DeleteImageSet	0.25 tps	1 tps
TagResource	10 tps	20 tps
ListTagsForResource	10 tps	20 tps
UntagResource	10 tps	20 tps
GetDICOMInstance*	50 tps	100 tps
GetDICOMInstanceMetadata*	50 tps	100 tps
GetDICOMInstanceFrames*	50 tps	100 tps
GetDICOMSeriesMetadata	50 tps	100 tps

\*DICOMweb 服務的代表

## AWS HealthImaging 像素資料驗證

在匯入期間，HealthImaging 透過檢查每個映像的無失真編碼和解碼狀態，提供內建像素資料驗證。此功能可確保使用 [HTJ2K 解碼程式庫解碼](#) 的影像一律符合匯入 HealthImaging 的原始 DICOM P10 影像。

- 當匯入任務在匯入之前擷取 DICOM P10 影像的原始像素品質狀態時，影像加入程序就會開始。使用 CRC32 演算法為每個影像產生唯一的不可變影像影格解析度檢查總和 (IFRC)。IFRC 檢查總和值會顯示在 `job-output-manifest.json` 中繼資料文件中。如需詳細資訊，請參閱 [了解匯入任務](#)。
- 將影像匯入 HealthImaging [資料存放區](#)並轉換為 [影像集](#)後，HTJ2K-encoded 的影像影格會立即解碼，並計算新的 IFRCs。HealthImaging 接著會將原始影像的完整解析度 IFRCs 與匯入影像影格的新 IFRCs 進行比較，以驗證準確性。
- 匯入任務輸出日誌 (`job-output-manifest.json`) 中會擷取對應的每個影像描述性錯誤條件，供您檢閱和驗證。

### 驗證像素資料

- 匯入醫療影像資料後，請檢視匯入任務輸出日誌 中擷取的每個影像集描述性成功（或錯誤條件）`job-output-manifest.json`。如需詳細資訊，請參閱 [了解匯入任務](#)。
- [影像集由中繼資料](#)和[影像影格](#)（像素資料）組成。影像集中繼資料包含關聯影像影格的相關資訊。使用 `GetImageSetMetadata`動作來取得映像集的中繼資料。如需詳細資訊，請參閱 [取得映像集中繼資料](#)。
- `PixelDataChecksumFromBaseToFullResolution` 包含完整解析度影像的 IFRC（檢查總和）。對於存放在原始傳輸語法 `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`（僅限二進位分段）中的影像，檢查總和會計算在原始影像上。對於存放在 HTJ2K Lossless with RPCL 中的影像，檢查總和會在解碼的全解析度影像上計算。如需詳細資訊，請參閱 [支援的傳輸語法](#)。

以下是 IFRC 的範例中繼資料輸出，這是匯入任務程序中產生並記錄至 的一部分 `job-output-manifest.json`。

```
"ImageFrames": [{}  
  "ID": "67890678906789012345123451234512",  
  "PixelDataChecksumFromBaseToFullResolution": [  
    {}  
      "Width": 512,
```

```
    "Height": 512,  
    "Checksum": 2510355201  
}  
]
```

### 對於存放在原始傳輸語法

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（僅限二進位分段）中的影像，MinPixelValue和MaxPixelValue 將無法使用。FrameSizeInBytes 指出原始影格的大小。

```
"PixelDataChecksumFromBaseToFullResolution": [  
    {"Width": 512, "Height": 512, "Checksum": 1379921327 }  
,  
    "MinPixelValue": null,  
    "MaxPixelValue": null,  
    "FrameSizeInBytes": 429
```

### 對於存放在 HTJ2K Lossless with RPCL 中的影像，FrameSizeInBytes 表示解碼影像影格的大小。

```
"PixelDataChecksumFromBaseToFullResolution": [  
    {"Width": 512, "Height": 512, "Checksum": 1379921327 }  
,  
    "MinPixelValue": 11,  
    "MaxPixelValue": 11,  
    "FrameSizeInBytes": 1652
```

4. 對於包含視訊的執行個體，HealthImaging 會執行輕量型轉碼器驗證，以驗證 DICOM 中繼資料中指定的傳輸語法符合視訊轉碼器。

HealthImaging 會使用 CRC32 演算法計算原始視訊物件上的 IFRC 檢查總和值。IFRC 檢查總和值會記錄到 job-output-manifest.json，並保留在 HealthImaging 中繼資料中。與存放在原始傳輸語法（如上所述）中的映像一樣，MinPixelValue和MaxPixelValue 將無法使用。FrameSizeInBytes 會指出原始影格的大小。

5. HealthImaging 會驗證像素資料、存取 GitHub 上的像素資料驗證程序，並遵循 檔案中 README.md 的指示來獨立驗證 HealthImaging 所使用的各種 HTJ2K 解碼程式庫的無失真映像處理。載入完整映像後，您可以計算端點上原始輸入資料的 IFRC，並將其與 HealthImaging 中繼資料中提供的 IFRC 值進行比較，以驗證像素資料。

## 適用於 AWS HealthImaging 的 HTJ2K 解碼程式庫

在匯入期間，有些傳輸語法會保留其原始編碼，而有些則預設會轉碼為高通量 JPEG 2000 (HTJ2K) 無損。HTJ2K 提供持續快速的映像顯示和 HTJ2K 進階功能的通用存取。由於某些影像影格是在匯入期間以 HTJ2K 編碼，因此在影像檢視器中檢視之前必須先解碼。如需判斷傳輸語法的資訊，請參閱[支援的傳輸語法](#)。

### Note

HTJ2K 在 [JPEG2000 標準 \(ISO/IEC 15444-15 : 2019\) 的第 15 部分](#)中定義。HTJ2K 保留 JPEG2000 的進階功能，例如解析度可擴展性、區域、並排、高位元深度、多頻道和色彩空間支援。

### 主題

- [HTJ2K 解碼程式庫](#)
- [影像檢視器](#)

## HTJ2K 解碼程式庫

根據您的程式設計語言，我們建議您使用下列解碼程式庫來解碼影像影格。

- [NVIDIA nvJPEG2000](#) – 商業 GPU 加速
- [卡卡杜軟體](#) – 商業、C++ 搭配 Java 和 .NET 繫結
- [OpenJPH](#) – 開放原始碼、C++ 和 WASM
- [OpenJPEG](#) – 開放原始碼、C/C++、Java
- [openjphpy](#) – 開放原始碼、Python
- [pylibjpeg-openjpeg](#) – 開放原始碼、Python

## 影像檢視器

您可以在解碼影像影格之後檢視影像影格。AWS HealthImaging API 動作支援各種開放原始碼映像檢視器，包括：

- [Open Health Imaging Foundation \(OHIF\)](#)
- [基石.js](#)

# AWS HealthImaging 範例專案

AWS HealthImaging 在 GitHub 上提供下列範例專案。

## 從現場部署到 AWS HealthImaging 的 DICOM 擷取

無 AWS 伺服器專案，用於部署從 DICOM DIMSE 來源 (PACS、VNA、CT 掃描器) 接收 DICOM 檔案的 IoT 邊緣解決方案，並將其存放在安全的 Amazon S3 儲存貯體中。解決方案會為資料庫中的 DICOM 檔案編製索引，並將要在 AWS HealthImaging 中匯入的每個 DICOM 系列排入佇列。它包含在邊緣執行的元件，由管理 [AWS IoT Greengrass](#)，以及在 AWS 雲端中執行的 DICOM 擷取管道。

## 並排層級標記 (TLM) Proxy

使用圖磚層級標記 (TLM) 從 AWS HealthImaging 擷取影像影格的 [AWS Cloud Development Kit \(AWS CDK\)](#) 專案，TLM 是高輸送量 JPEG 2000 (HTJ2K) 的功能。這會導致較低解析度影像的擷取時間更快。潛在的工作流程包括產生縮圖和逐步載入影像。

## Amazon CloudFront 交付

使用 HTTPS 端點建立 [Amazon CloudFront](#) 分佈的無 AWS 伺服器專案，該端點快取（使用 GET）並從邊緣交付影像影格。根據預設，端點會使用 Amazon Cognito JSON Web 字符 (JWT) 驗證請求。驗證和請求簽署都是在邊緣使用 [Lambda@Edge](#) 完成。此服務是 Amazon CloudFront 的一項功能，可讓您在更接近應用程式使用者的地方執行程式碼，進而改善效能並降低延遲。沒有要管理的基礎設施。

## AWS HealthImaging Viewer UI

使用後端身分驗證部署前端 UI 的 [AWS Amplify](#) 專案，您可以使用此專案來檢視影像集中繼資料屬性，以及使用漸進式解碼儲存在 AWS HealthImaging 中的影像影格（像素資料）。您可以選擇性地整合上面的並排層級標記 (TLM) Proxy 和/或 Amazon CloudFront Delivery 專案，以使用替代方法載入影像影格。

## AWS HealthImaging DICOMweb Proxy

以 Python 為基礎的專案，可在 HealthImaging 資料存放區上啟用 DICOMweb WADO-RS 和 QIDO-RS 端點，以支援 Web 型醫療影像檢視器和其他 DICOMweb 相容應用程式。

### Note

此專案不會使用 HealthImaging 中描述的 DICOMweb APIs 表示法 [搭配 AWS HealthImaging 使用 DICOMweb](#)。

若要檢視其他範例專案，請參閱 GitHub 上的 [AWS HealthImaging 範例](#)。

## 將此服務與 AWS SDK 搭配使用

AWS 軟體開發套件 SDKs) 適用於許多熱門的程式設計語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

SDK 文件	代碼範例
<a href="#">適用於 C++ 的 AWS SDK</a>	<a href="#">適用於 C++ 的 AWS SDK 程式碼範例</a>
<a href="#">AWS CLI</a>	<a href="#">AWS CLI 程式碼範例</a>
<a href="#">適用於 Go 的 AWS SDK</a>	<a href="#">適用於 Go 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 Java 的 AWS SDK</a>	<a href="#">適用於 Java 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 JavaScript 的 AWS SDK</a>	<a href="#">適用於 JavaScript 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 Kotlin 的 AWS SDK</a>	<a href="#">適用於 Kotlin 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 .NET 的 AWS SDK</a>	<a href="#">適用於 .NET 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 PHP 的 AWS SDK</a>	<a href="#">適用於 PHP 的 AWS SDK 程式碼範例</a>
<a href="#">AWS Tools for PowerShell</a>	<a href="#">適用於 PowerShell 的工具程式碼範例</a>
<a href="#">適用於 Python (Boto3) 的 AWS SDK</a>	<a href="#">適用於 Python (Boto3) 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 Ruby 的 AWS SDK</a>	<a href="#">適用於 Ruby 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 Rust 的 AWS SDK</a>	<a href="#">適用於 Rust 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 SAP ABAP 的 AWS SDK</a>	<a href="#">適用於 SAP ABAP 的 AWS SDK 程式碼範例</a>
<a href="#">適用於 Swift 的 AWS SDK</a>	<a href="#">適用於 Swift 的 AWS SDK 程式碼範例</a>

## 可用性範例

找不到所需的內容嗎？請使用本頁面底部的提供意見回饋連結申請程式碼範例。

# AWS HealthImaging 版本

下表顯示何時發佈 AWS HealthImaging 服務和文件的功能和更新。如需版本的詳細資訊，請參閱連結主題。

變更	描述	日期
<a href="#">自動資料組織、DICOMweb</a> <a href="#">QIDO-RS 搜尋和 DICOMweb</a> <a href="#">WADO-RS 增強功能</a>	HealthImaging 會根據 DICOM 標準的患者、研究和序列層級階層，在匯入時自動整理 DICOM P10 資料。如需詳細資訊，請參閱 <a href="#">了解匯入任務</a> 。HealthImaging 根據 DICOMweb QIDO-RS 標準支援豐富的搜尋。如需詳細資訊，請參閱在 <a href="#">HealthImaging 中搜尋 DICOM 資料</a> 。HealthImaging 支援透過單一 API 動作擷取序列中所有 DICOM 執行個體的中繼資料，如從 <a href="#">從 HealthImaging 取得 DICOM 系列中繼資料</a> 所述。	2025 年 5 月 22 日
<a href="#">影像集建立使用較少的 DICOM 元素</a>	HealthImaging 可減少將傳入 DICOM P10 物件分組到影像集時所使用的元素數量。如需詳細資訊，請參閱 <a href="#">什麼是映像集？</a> 。	2025 年 1 月 27 日
<a href="#">StartDICOMImportJob 的失真支援</a>	HealthImaging 支援以原始格式匯入和儲存 DICOM 遺失檔案 (1.2.840.10008.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50) 和二進位分段檔案。如需詳細資訊，請參閱 <a href="#">支援的傳輸語法</a> 。	2024 年 11 月 1 日

<a href="#"><u>DICOMweb 擷取 APIs失真支援</u></a>	HealthImaging 支援以原始格式或明確 VR Little Endian (1.2.840.1.2.4.203、1.2.840.10008.1.2.4.91、1.2.840.10008.1.2.4.50 和 1.2.840.10008.1.2.1 ( 僅限二進位分段 ) 儲存的影像和執行個體。如需詳細資訊，請參閱 <a href="#">支援的傳輸語法和擷取 DICOM 資料</a> 。	2024 年 11 月 1 日
<a href="#"><u>更快速地匯入數位邏輯</u></a>	HealthImaging 支援高達 6 倍的 DICOM 數位邏輯 (WSI) 匯入任務。	2024 年 11 月 1 日
<a href="#"><u>二進位分段支援</u></a>	HealthImaging 支援 DICOM 二進位分段檔案的擷取和擷取。如需詳細資訊，請參閱 <a href="#">支援的傳輸語法</a> 。	2024 年 11 月 1 日
<a href="#"><u>還原至先前的映像集版本 ID</u></a>	HealthImaging 提供 <code>revertToVersionId</code> 參數，以還原至先前的映像集版本 ID。如需詳細資訊，請參閱 AWS HealthImaging API 參考 <a href="#">revertToVersionId</a> 中的。	2024 年 7 月 24 日

## 強制修改映像集的功能

HealthImaging 提供具有選用 forced 請求參數的 Overrides 資料類型。即使病患、檢查或序列層級中繼資料不相符，設定此參數也會強制 UpdateImageSetMetadata 和 CopyImageSet 動作。如需詳細資訊，請參閱 AWS HealthImaging API 參考中的 [覆寫](#)。

2024 年 7 月 24 日

- UpdateImageSetMetadata 強制功能 - HealthImaging 推出選用的 force 請求參數來更新下列屬性：
  - Tag.StudyInstanceUID, Tag.SeriesInstanceUID, Tag.SOPInstanceUID, 和 Tag.StudyID
  - 新增、移除或更新執行個體層級私有 DICOM 資料元素

如需詳細資訊，請參閱 AWS HealthImaging API 參考中的 [UpdateImageSetMetadata](#)。

- CopyImageSet 強制功能 - HealthImaging 引入用於複製映像集的選用 force 請求參數。即使 sourceImageSet 和之間的病患、檢查或序列層級中繼資料不相符，設定此參數也會強制 CopyImageSet 動作 destinationImageSe

t。在這些情況下，不一致的中繼資料在 sourceImageSet 中保持不變 destinationImageSet。如需詳細資訊，請參閱《AWS HealthImaging API 參考》中的 [CopyImageSet](#)。

### [複製 SOP 執行個體的子集](#)

HealthImaging 會增強 CopyImageSet 動作，因此您可以從挑選一或多個 SOP 執行個體 sourceImageSet，以複製到 destinationImageSet。如需詳細資訊，請參閱 [複製映像集](#)。

2024 年 7 月 24 日

### [GetDICOMInstanceMetadata](#) 用於傳回 DICOM 執行個體中繼資料

HealthImaging 提供 GetDICOMInstanceMetadata API 來傳回 DICOM Part 10 中繼資料 (.json 檔案)。如需詳細資訊，請參閱 [取得執行個體中繼資料](#)。

2024 年 7 月 11 日

### [GetDICOMInstanceFrames](#) 用於傳回 DICOM 執行個體影格（像素資料）

HealthImaging 提供 GetDICOMInstanceFrames API 來傳回 DICOM Part 10 影格 (multipart 請求)。如需詳細資訊，請參閱 [取得執行個體影格](#)。

2024 年 7 月 11 日

## 增強對非標準 DICOM 資料匯入的支援

HealthImaging 支援包含偏離 DICOM 標準的資料匯入。如需詳細資訊，請參閱 [DICOM 元素限制](#) 條件。

2024 年 6 月 28 日

- 下列 DICOM 資料元素的長度上限為 256 個字元：
  - Patient's Name (0010,0010)
  - Patient ID (0010,0020)
  - Accession Number (0008,0050)
  - Study Instance UID、Series Instance UID、Treatment Session UID、Manufacturer's Device Class UIDDevice UID 和允許下列語法變化Acquisition UID：
  - 任何 UID 的第一個元素可以是零
  - UIDs的開頭可以是一或多個前導零
  - UIDs 長度上限為 256 個字元

## 事件通知

HealthImaging 與 Amazon EventBridge 整合，以支援事件驅動型應用程式。如需詳細資訊，請參閱《[使用 EventBridge](#)》。

2024 年 6 月 5 日

<a href="#"><u>GetDICOMInstance 用於傳回 DICOM 執行個體資料</u></a>	HealthImaging 提供傳回 DICOM Part 10 執行個體資料 (.dcm 檔案) GetDICOMInstance 的服務。如需詳細資訊，請參閱 <a href="#">取得執行個體</a> 。	2024 年 5 月 15 日
<a href="#"><u>跨帳戶匯入</u></a>	HealthImaging 支援從位於其他支援區域的 Amazon S3 儲存貯體匯入資料。如需詳細資訊，請參閱 <a href="#">跨帳戶匯入</a> 。	2024 年 5 月 15 日
<a href="#"><u>影像集的搜尋增強功能</u></a>	HealthImaging SearchImageSets 動作支援下列搜尋增強功能。如需詳細資訊，請參閱 <a href="#">搜尋映像集</a> 。 <ul style="list-style-type: none"><li>• 在 UpdatedAt 和上搜尋的其他支援 SeriesInstanceUID</li><li>• 在開始時間和結束時間之間搜尋</li><li>• 依 Ascending 或排序搜尋結果 Descending</li><li>• DICOM 系列參數會在回應中傳回</li></ul>	2024 年 4 月 3 日
<a href="#"><u>已增加匯入的檔案大小上限</u></a>	HealthImaging 支援匯入任務中每個 DICOM P10 檔案的檔案大小上限為 4 GB。如需詳細資訊，請參閱 <a href="#">Service Quotas</a> 。	2024 年 3 月 6 日

## [傳輸 JPEG 無失真和 HTJ2K 的語法](#)

HealthImaging 支援下列任務匯入的傳輸語法。如需詳細資訊，請參閱[支援的傳輸語法](#)。

2024 年 2 月 16 日

- 1.2.840.10008.1.2.4.57 — JPEG 無損非階層式（程序 14）
- 1.2.840.10008.1.2.4.201 — 高輸送量 JPEG 2000 影像壓縮（僅限無損）
- 1.2.840.10008.1.2.4.202 — 具有 RPCL 選項影像壓縮的高輸送量 JPEG 2000（僅限無損）
- 1.2.840.10008.1.2.4.203 — 高輸送量 JPEG 2000 影像壓縮

## [已測試的程式碼範例](#)

HealthImaging 文件提供適用於 Python、JavaScript、Java 和 C++ 的 AWS CLI 和 AWS SDKs 的測試程式碼範例。如需詳細資訊，請參閱[程式碼範例](#)。

2023 年 12 月 19 日

## [增加的匯入檔案數目上限](#)

HealthImaging 最多支援單一匯入任務 5,000 個檔案。如需詳細資訊，請參閱[Service Quotas](#)。

2023 年 12 月 19 日

## [匯入的巢狀資料夾](#)

HealthImaging 支援單一匯入任務最多 10,000 個巢狀資料夾。如需詳細資訊，請參閱[服務配額](#)。

2023 年 12 月 1 日

更快速匯入

HealthImaging 在所有支援的區域中提供快 20X 的匯入。如需詳細資訊，請參閱[服務端點](#)。

AWS CloudFormation 支援

HealthImaging 支援基礎設施即程式碼 (IaC) 來佈建資料存放區。如需詳細資訊，請參閱[使用建立 HealthImaging 資源 AWS CloudFormation](#)。

一般可用性

AWS HealthImaging 可供美國東部（維吉尼亞北部）、美國西部（奧勒岡）、歐洲（愛爾蘭）和亞太區域（雪梨）區域的所有客戶使用。如需詳細資訊，請參閱[服務端點](#)。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。