aws

開發人員指南

AWS HealthImaging

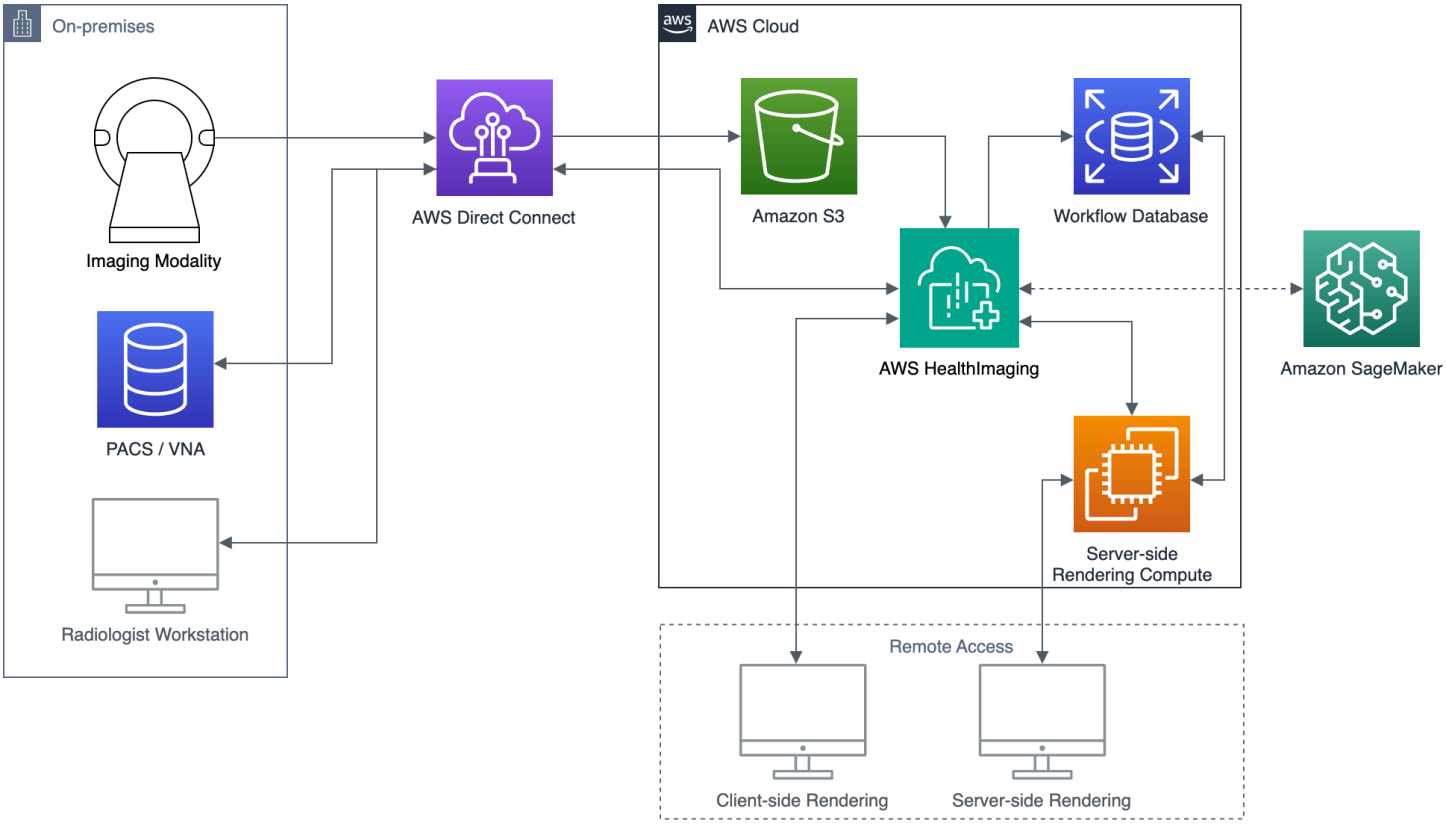
AWS HealthImaging: 開發人員指南

Table of Contents

什麼是 AWS HealthImaging？

AWS HealthImaging 是符合 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在雲端以 PB 規模存放、分析和共用醫療影像。 HealthImaging用例包括：

- 企業影像 — 直接從 AWS 雲端儲存和串流您的醫療影像資料，同時保留低延遲效能和高可用性。
- 長期影像存檔 — 節省長期影像封存的成本，同時維持低於一秒的影像擷取存取。
- AI/ML 開發 — 透過其他工具和服務的支援，在影像封存上執行人工智慧和機器學習 (AI/ML) 推論。
- 多模態分析 — 將您的臨床影像資料與 AWS HealthLake （健康資料）和 AWS HealthOmics （組學資料）相結合，以提供精準醫學的見解。



AWS 可 HealthImaging 讓您存取影像資料 (例如 X-Ray、CT、MRI、超音波)，因此雲端內建的醫療影像應用程式能夠達到之前只有現場部署才能實現的效能。有了 HealthImaging，您可以從中 AWS 雲端的每個醫療影像的單一授權副本大規模執行醫學影像應用程式，藉此降低基礎架構成本。

主題

- 重要通知

重要通知

AWS 不 HealthImaging 是專業醫療建議、診斷或治療的替代品，並非用於治愈、治療、減輕、預防或診斷任何疾病或健康狀況。您有責任在 AWS 的任何使用過程中進行人工審查 HealthImaging，包括與任何旨在通知臨床決策的第三方產品相關聯。只有 HealthImaging 在經過訓練的醫療專業人員進行審查後，才應將 AWS 用於病患照護或臨床案例。

AWS 的功能 HealthImaging

AWS HealthImaging 提供下列功能。

對開發者友善的 DICOM 中繼資料

AWS 以適合開發人員的格式傳回 DICOM 中繼資料，以 HealthImaging 簡化應用程式開發。匯入影像資料後，可以使用人性化的關鍵字存取個別中繼資料屬性，而非陌生的群組/元素十六進位數字。病患、研究和系列層級的 DICOM 元素都經過標準化，因此應用程式開發人員無需處理 SOP 執行個體之間的不一致性。此外，中繼資料屬性值也可以在原生執行階段類型中直接存取。

SIMD 加速影像解碼

AWS HealthImaging 傳回使用高輸送量 JPEG 2000 (HTJ2K) (一種進階影像壓縮轉碼器) 編碼的影像框 (像素資料)。HTJ2K 利用現代處理器上的單指令多重資料 (SIMD)，提供全新等級的效能。HTJ2K 的速度比 JPEG2000 快，速度至少是所有其他 DICOM 傳輸語法的兩倍。WASM-SIMD 可以用來為零佔用空間的網頁觀看者帶來這種極致的速度。

像素數據驗證

AWS HealthImaging 透過在匯入期間檢查每個影像的無失真編碼和解碼狀態，提供內建的像素資料驗證。如需詳細資訊，請參閱 像素數據驗證。

領先業界的

AWS 憑藉其高效的中繼資料編碼、無失真壓縮和漸進式解析度資料存取，為影像載入效能 HealthImaging 樹立了新的標準。高效率的中繼資料編碼可讓影像檢視器和 AI 演算法瞭解 DICOM

研究的內容，而無需載入影像資料。由於先進的圖像壓縮，圖像加載速度更快，而不會影響圖像質量。漸進式解析度可讓縮圖、感興趣的區域和低解析度行動裝置的影像載入速度更快。

可擴充的 DICOM 匯入

AWS HealthImaging 匯入會利用現代雲端原生技術 parallel 匯入多個 DICOM 研究。歷史存檔可以快速匯入，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，請參閱迪克姆支持。

相關 AWS 服務

AWS 具有與其他 AWS 服務緊密整合的 HealthImaging 功能。對以下服務的了解對於充分利用很有用 HealthImaging。

- AWS Identity and Access Management— 您可以使用 IAM 安全地管理身分識別和 HealthImaging 資源存取權。
- 亞馬遜簡單儲存服務 — 您可以使用 Amazon S3 做為將 HealthImaging DICOM 資料匯入的暫存區域。
- Amazon CloudWatch — 您可 CloudWatch 以用來觀察和監控 HealthImaging 資源。
- AWS CloudTrail— 用 CloudTrail 於跟踪用 HealthImaging 戶活動和 API 使用情況。
- AWS CloudFormation— 您可 AWS CloudFormation 以用來實作基礎結構作為程式碼 (IaC) 範本，以在 HealthImaging.
- AWS PrivateLink— 您可以使用 Amazon VPC 在 HealthImaging 和 Amazon 虛擬私有雲之間建立連線，而不會將資料暴露到網際網路。

存取 AWS HealthImaging

您可以使 HealthImaging 用 AWS Command Line Interface 和 AWS 開發 AWS Management Console 套件存取 AWS。本指南提供和 AWS SDK 的程序指示 AWS Management Console 和程式碼範例。
AWS CLI

AWS Management Console

AWS Management Console 提供用於管理 HealthImaging 及其相關資源的網頁式使用者介面。如果您已註冊 AWS 帳戶，則可以登入HealthImaging 主機。

AWS Command Line Interface (AWS CLI)

提 AWS CLI 供多種 AWS 產品的指令，並在視窗、Mac 和 Linux 上受到支援。如需詳細資訊，請參閱 AWS Command Line Interface 使用者指南。

AWS 開發套件

AWS SDK 為軟體開發人員提供程式庫、程式碼範例和其他資源。這些程式庫提供可自動化工作的基本功能，例如加密簽署要求、重試要求，以及處理錯誤回應。如需詳細資訊，請參閱在 AWS上建置的工具。

HTTP 請求

您可以使用 HTTP 要求呼叫 HealthImaging 動作，但必須根據所使用的動作類型指定不同的端點。如需詳細資訊，請參閱 HTTP 要求支援的 API 動作。

HIPAA 資格和資料安全性

此為 HIPAA 合格服務。如需有關 AWS《1996 年美國 Health 保險流通與責任法案》(HIPAA)，以及使用 AWS 服務來處理、儲存和傳輸受保護的健康資訊 (PHI) 的詳細資訊，請參閱 HIP AA 概觀。

HealthImaging 包含 PHI 和個人識別資訊 (PII) 的連線必須經過加密。根據預設，所有連線都會透過 TLS HealthImaging 使用 HTTPS。 HealthImaging 儲存加密的客戶內容，並根據AWS 共同責任模型進行操作。

如需符合性的相關資訊，請參閱適用於 AWS 的合規驗證 HealthImaging。

定價

HealthImaging 透過智慧型分層，協助您將臨床資料的生命週期管理自動化。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。若要估算成本，請使用 AWS HealthImaging 定價計算器。

開始使用 AWS HealthImaging

若要開始使用 AWS HealthImaging，請設定 AWS 帳戶並建立使用 AWS Identity and Access Management 者。若要使用AWS CLI或 AWS SDK，您必須安裝並設定它們。

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

主題

- AWS HealthImaging 概念
- 設定 AWS HealthImaging
- AWS HealthImaging 教學課程

AWS HealthImaging 概念

以下術語和概念是您對 AWS 的理解和使用的核心 HealthImaging。

概念

- 資料存放區
- 影像集
- 中繼資料
- 影像框

資料存放區

資料倉庫是位於單一醫學影像資料的儲存庫AWS 區域。一個AWS帳戶可以有零個或多個資料存放區。資料倉庫具有自己的AWS KMS加密金鑰，因此一個資料倉庫中的資料可以在實體和邏輯上與其他資料存放區中的資料隔離。資料存放區使用 IAM 角色、許可和以屬性為基礎的存取控制來支援存取控制。

如需詳細資訊，請參閱 管理資料倉庫 及 了解儲存層。

影像集

影像集是定義用於最佳化相關醫學影像資料的抽象分組機制的AWS概念。將 DICOM P10 影像資料匯入 AWS 資 HealthImaging 料存放區時，它會轉換為由中繼資料和影像畫面 (像素資料) 組成的映像集。匯入 DICOM P10 資料會產生包含相同 DICOM 系列中之一或多個服務-物件配對 (SOP) 執行個體的 DICOM 中繼資料和影像框架的映像集。

如需詳細資訊，請參閱 匯入影像資料 及 瞭解影像集。

中繼資料

中繼資料是存在於影像集內的非像素屬性。對於 DICOM，這包括患者人口統計信息，程序詳細信息以及其他獲取特定參數。AWS 會將影像集 HealthImaging 分成中繼資料和影像框架 (像素資料)，以便應用程式可以快速存取。這對於不需要像素資料的影像檢視器、分析和 AI/ML 使用案例很有幫助。DICOM 資料會在「患者」、「研究」和「系列」層級標準化，從而消除不一致之處。這樣可簡化資料的使用、提高安全性並改善存取效能。

如需詳細資訊，請參閱 取得影像集中繼資料 及 元数据规范化。

影像框

圖像幀是存在於圖像集中的像素數據，用於組成 2D 醫學圖像。在匯入期間，AWS 會以高輸送量 JPEG (HTJ2K) 對所有影像畫面進行 HealthImaging 編碼。因此，在檢視之前，必須先解碼影像框。

如需更多詳細資訊，請參閱 取得影像集像素資料 及 HTJ2K 解碼程式庫。

設定 AWS HealthImaging

在使用 AWS 之前，您必須先設定 AWS 環境 HealthImaging。下列主題是下一節中自學課程的先決條件。

主題

- 註冊一個 AWS 帳戶
- 建立具有管理權限的使用者
- 建立 S3 儲存貯體
- 建立資料存放區
- 建立具有 HealthImaging 完整存取權限的 IAM 使用者
- 建立用於匯入的 IAM 角色
- 安裝 AWS CLI （可選）

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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

1. 在 IAM 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。
2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》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 輸入儲存貯體會存放要匯入的 DICOM P10 資料，並從此儲存貯體 HealthImaging 讀取資料。Amazon S3 輸出儲存貯體會存放匯入任務的處理結果並 HealthImaging寫入此儲存貯體。如需此的視覺化表示方式，請參閱的圖表瞭解匯入工作。

> **Note**
>
> 由於 AWS Identity and Access Management (IAM) 政策的限制，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。

針對本指南的目的，我們在要匯入的 IAM 角色中指定下列 Amazon S3 輸入和輸出儲存貯體。

- 輸入桶：`arn:aws:s3:::medical-imaging-dicom-input`
- 輸出桶：`arn:aws:s3:::medical-imaging-output`

如需其他資訊，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 資 HealthImaging 料存放區會保留轉換後的 DICOM P10 檔案的結果，這些檔案稱為映像集。如需此的視覺化表示方式，請參閱的圖表瞭解匯入工作。

> **Tip**
>
> 建立資料倉庫時會產生 A。`datastoreID`完成本節稍後的匯入`datastoreID`trust relationship時，您必須使用。

若要建立資料倉庫，請參閱建立資料倉庫。

建立具有 HealthImaging 完整存取權限的 IAM 使用者

> **最佳實務**
>
> 我們建議您針對不同需求 (例如匯入、資料存取和資料管理) 建立個別的 IAM 使用者。這與 AWS Well-Architected 的框架中授予最少權限訪問一致。
> 基於下一節教學課程的目的，您將使用單一 IAM 使用者。

建立 IAM 使用者

1. 按照 IAM 使用者指南中有關在 AWS 帳戶中建立 IAM 使用者的說明進行操作。考慮為澄清目的命名用戶`ahiadmin`（或類似用戶）。
2. 將受`AWSHealthImagingFullAccess`管政策指派給 IAM 使用者。如需詳細資訊，請參閱 AWS 受管理策略： AWSHealthImagingFullAccess。

   > **Note**
   >
   > IAM 許可可可可可可縮小。如需詳細資訊，請參閱 AWS適用於 AWS 的受管政策 HealthImaging。

建立用於匯入的 IAM 角色

> **Note**
>
> 以下說明是一個 AWS Identity and Access Management (IAM) 角色，該角色授予 Amazon S3 儲存貯體的讀取和寫入存取權以匯入 DICOM 資料。雖然下一節的教學課程需要此角色，但我們建議您使用新增 IAM 許可給使用者、群組和角色AWS適用於 AWS 的受管政策 HealthImaging，因為這些權限比自己編寫政策更容易使用。

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 角色必須附加至使用者政策，該政策授與用於讀取 DICOM P10 資料和存放匯入任務處理結果之 Amazon S3 儲存貯體的存取權。此外，還必須為其指派信任關係 (政策)，才能 HealthImaging 讓 AWS 擔任該角色。

若要建立 IAM 角色以進行匯入

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**
   >
   > 針對本指南的目的，中的程式碼範例會開始匯入工作參考 `ImportJobDataAccessRole` IAM 角色。

2. 將 IAM 權限政策附加到 IAM 角色。此權限政策授予對 Amazon S3 輸入和輸出儲存貯體的存取權。將下列權限政策附加到 IAM 角色`ImportJobDataAccessRole`。

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

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

3. 將下列信任關係 (政策) 附加到 ImportJobDataAccessRole IAM 角色。信任原則需要datastoreId您完成區段時所產生的建立資料存放區。本主題後面的教學課程假設您使用一個 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。

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

```
                }
            }
        }
    ]
}
```

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

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（可選）

如果您使用的是，則需要執行下列程序 AWS Command Line Interface。如果您使用的是 AWS Management Console 或 AWS SDK，則可以略過下列程序。

若要設定 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，為管理員新增具名的設定檔。您可以在執行 AWS CLI 指令時使用此設定檔。在最低權限的安全性原則下，建議您建立個別的 IAM 角色，該角色具有特定於所執行任務的特權。如需有關具名設定檔的詳細資訊，請參閱AWS Command Line Interface 使用指南中的組態和認證檔案設定。

   ```
   [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`命令驗證設置。

   ```
   aws medical-imaging help
   ```

   如果設定 AWS CLI 正確，您會看到 AWS 的簡短說明 HealthImaging和可用命令的清單。

AWS HealthImaging 教學課程

目標

本教學的目標是將 DICOM P10 檔案匯入 AWS HealthImaging 資料存放區，並將其轉換為由中繼資料和影像畫面 (像素資料) 組成的映像集。匯入 DICOM 資料之後，您可以根據您的存取偏好設定存取影像集、中繼資料和影像框。 HealthImaging

必要條件

中列示的所有程序設定都需要完成此自學課程。

教學步驟

1. 開始匯入工作
2. 取得匯入工作屬性
3. 搜尋影像集
4. 取得影像集屬性
5. 取得影像集中繼資料
6. 獲取圖像集像素數據
7. 刪除資料倉庫

使用 AWS 管理資料存放區 HealthImaging

使用 AWS HealthImaging，您可以為醫療影像資源建立和管理資料存放區。下列主題描述如何使用 HealthImaging 動作來建立、描述、列出和刪除使用 AWS Management Console AWS CLI、和 AWS SDK 的資料倉庫。

> **Note**
>
> 本章的最後一個主題是關於瞭解儲存層。將醫療影像資料匯入資料存放區後，它會根據時間和使用情況自動在兩個儲存層之間移動。儲存層有不同的定價層級，因此瞭解階層移動程序以及為計費目的而識別的 HealthImaging 資源非常重要。

主題

建立資料倉庫

您可以使用此`CreateDatastore`動作建立用於匯入 DICOM P10 檔案的 AWS HealthImaging 資料存放區。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請`CreateDatastore`參閱 AWS HealthImaging API 參考中的。

> **重要**
>
> 請勿使用受保護的健康資訊 (PHI)、個人識別資訊 (PII) 或其他機密或敏感資訊來命名資料儲存區。

建立資料倉庫的步驟

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台「建立資料倉庫」頁面。
2. 在「詳細資料」下，對於「資料倉庫名稱」，輸入資料倉庫的名稱。
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 中的資料保護 HealthImaging。
4. 在「標籤-可選」下，您可以在建立資料倉庫時將標籤新增至資料倉庫。如需詳細資訊，請參閱 標記資源。
5. 選擇 [建立資料倉庫]。

AWS CLI 和軟體開發套件

Bash

AWS CLI 與 bash 腳本

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

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

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

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

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

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

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

  local error_code=${?}

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

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

  echo "$response"

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CreateDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

建立資料倉庫的步驟

下列create-datastore程式碼範例會建立名稱的資料存放區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倉庫。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CreateDatastore中的。

Java

適用於 Java 2.x 的 SDK

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

       return "";
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考CreateDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考CreateDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件CreateDatastore中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

取得資料倉庫屬性

您可以使用GetDatastore動作擷取 AWS HealthImaging 資料存放區屬性。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請GetDatastore參閱 AWS HealthImaging API 參考中的。

取得資料倉庫性質的步驟

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   資料倉庫詳細資訊頁面隨即開啟。在「詳細資料」區段下，所有資料倉庫性質均可用。若要檢視關聯的影像集、匯入和標籤，請選擇適用的索引標籤。

AWS CLI 和軟體開發套件

Bash

AWS CLI 使用 Bash 腳本

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

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

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

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

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

 local response

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

```
  )
  error_code=${?}

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

  echo "$response"

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

取得資料倉庫性質的步驟

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GetDatastore中的。

Java

適用於 Java 2.x 的 SDK

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetDatastore中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

列出資料倉庫

您可以使用此`ListDatastores`動作列出 AWS 中可用的資料存放區 HealthImaging。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請`ListDatastores`參閱 AWS HealthImaging API 參考中的。

列示資料倉庫的步驟

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

- 開啟 HealthImaging 主控台 [資料存放區] 頁面。

  所有資料倉庫都列在「資料存放區」區段下。

AWS CLI 和軟體開發套件

Bash

AWS CLI 使用 Bash 腳本

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

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

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

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

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

  echo "$response"

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

列示資料倉庫的步驟

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ListDatastores中的。

Java

適用於 Java 2.x 的 SDK

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

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

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

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

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

  return datastoreSummaries;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return datastore_summaries
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListDatastores中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

刪除資料倉庫

您可以使用DeleteDatastore動作刪除 AWS HealthImaging 資料存放區。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請DeleteDatastore參閱 AWS HealthImaging API 參考中的。

> **Note**
>
> 刪除資料倉庫之前，必須先刪除其中的所有影像集。如需詳細資訊，請參閱 刪除影像集。

刪除資料倉庫的步驟

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。

2. 選擇資料倉庫。
3. 選擇刪除。

   將開啟「刪除資料倉庫」頁面。
4. 若要確認刪除資料倉庫，請在文字輸入欄位中輸入資料倉庫名稱。
5. 選擇「刪除資料倉庫」。

AWS CLI 和軟體開發套件

Bash

AWS CLI 與 bash 腳本

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

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

  # bashsupport disable=BP5008
  function usage() {
    echo "function imaging_delete_datastore"
```

```
    echo "Deletes an AWS HealthImaging data store."
    echo "  -i datastore_id - The ID of the data store."
    echo ""
  }

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

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

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

  local error_code=${?}

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

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DeleteDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

刪除資料倉庫的步驟

下列delete-datastore程式碼範例會刪除資料倉庫。

```
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倉庫。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DeleteDatastore中的。

Java

適用於 Java 2.x 的 SDK

```
   public static void deleteMedicalImagingDatastore(MedicalImagingClient
medicalImagingClient,
           String datastoreID) {
       try {
           DeleteDatastoreRequest datastoreRequest =
DeleteDatastoreRequest.builder()
                   .datastoreId(datastoreID)
```

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DeleteDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

```
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteDatastore中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

了解儲存層

AWS HealthImaging 使用智慧型分層進行自動化臨床生命週期管理。這為新的或使用中的資料和長期存檔資料提供絕佳的效能和價格，而且零摩擦。 HealthImaging 使用下列層級計費每月每 GB 的儲存體費用。

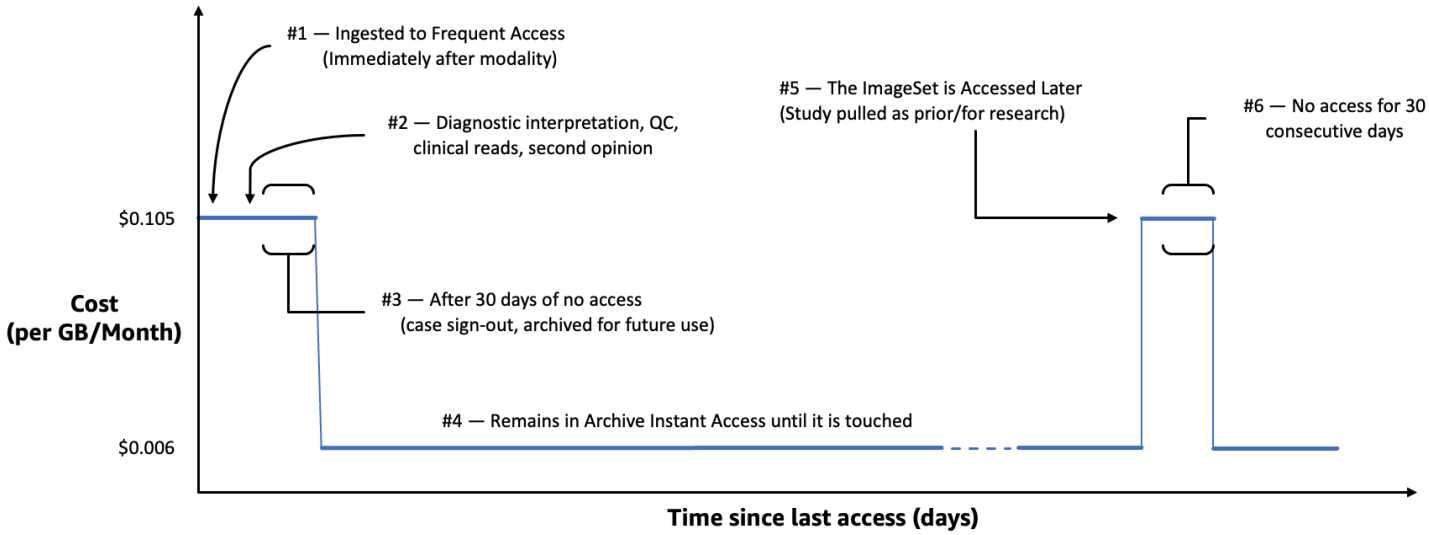
- 頻繁存取層 — 經常存取資料的層級。
- 封存即時存取層 — 封存資料的層級。

> **Note**
>
> 頻繁存取和封存即時存取層之間沒有效能差異。智慧型分層會套用至特定影像集 API 動作。智慧型分層無法辨識資料存放、匯入和標記 API 動作。層之間的移動是根據 API 使用情況自動進行的，並在下一節中說明。

階層移動如何工作？

- 匯入之後，影像集會從頻繁存取層開始。
- 連續 30 天沒有接觸後，映像集會自動移至封存即時存取層。
- 封存即時存取層中的映像集只有在觸碰後才會移回頻繁存取層。

下圖提供 HealthImaging 智慧型分層程序的概觀。



什麼被認為是觸摸？

觸控是透過、或 AWS SDK 存取的特定 API AWS Management ConsoleAWS CLI，並在下列情況發生：

1. 已建立新影像集 (`StartDICOMImportJob`或`CopyImageSet`)
2. 影像集已更新 (`UpdateImageSetMetadata`或`CopyImageSet`)
3. 讀取影像集關聯的中繼資料或影像框 (像素資料) (`GetImageSetMetaData`或`GetImageFrame`)

下列 HealthImaging API 動作會導致接觸，並將影像集從封存即時存取層移至頻繁存取層。

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

> **Note**
>
> 雖然無法使用`UpdateImageSetMetadata`動作刪除影像框 (像素資料)，但仍會計入計費用途。

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頻繁存取層。

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

使用 AWS 匯入影像資料 HealthImaging

匯入是將您的醫療影像資料從 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，然後再將 DICOM P10 檔案轉換為由中繼資料和影像畫面 (像素資料) 組成的映像集。

> **Tip**
>
> 熟悉完成之後 HealthImaging，我們建議您造訪以使用我們的匯入和檢視專案AWS HealthImaging 範例專案來取得實作快速啟動。

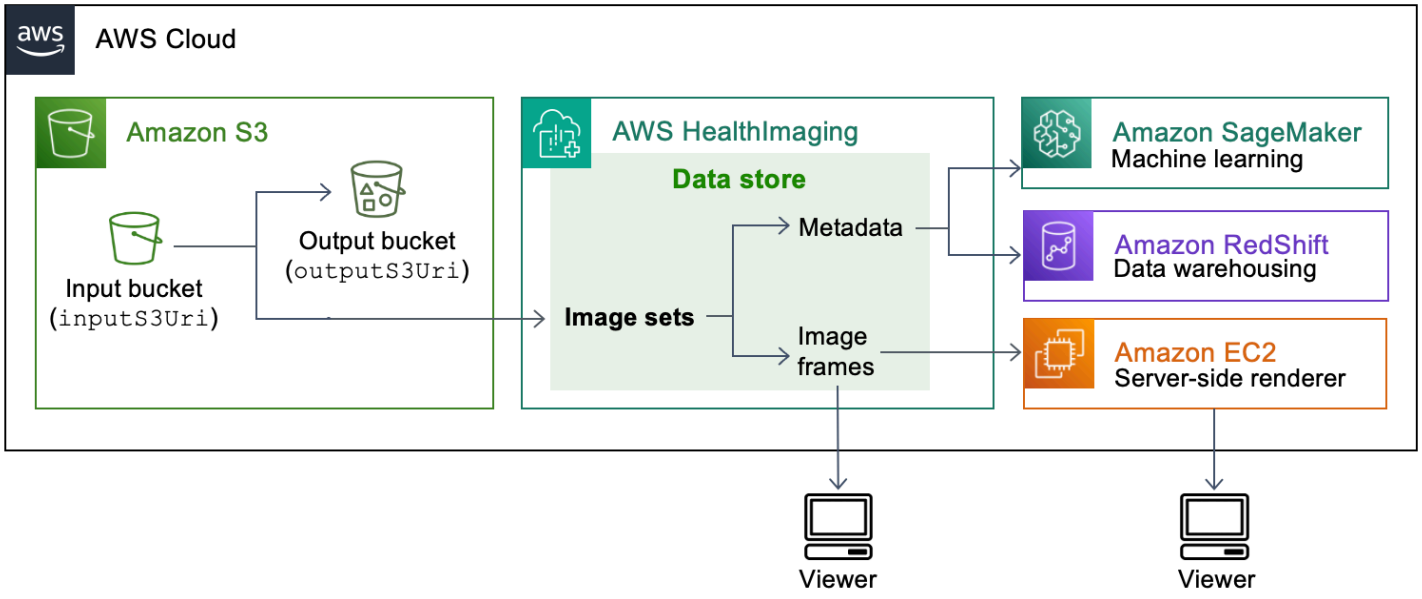
下列主題說明如何使用AWS Management Console、AWS CLI和 AWS SDK 將醫療影像 HealthImaging 資料匯入資料存放區。

主題

瞭解匯入工作

在 AWS 中建立資料存放區後 HealthImaging，您必須將您的醫療影像資料從 Amazon S3 輸入儲存貯體匯入資料存放區，才能建立映像集。您可以使用 AWS Management Console AWS CLI、和 AWS SDK 來啟動、描述和列出匯入工作。

下圖提供如何將 DICOM 資料 HealthImaging 匯入資料倉庫並將其轉換為影像集的概觀。匯入任務處理結果存放在 Amazon S3 輸出儲存貯體 (`outputS3Uri`) 中，而影像集則存放在 AWS HealthImaging 資料存放區中。



將您的醫療影像檔案從 Amazon S3 匯入 AWS HealthImaging 資料存放區時，請記住以下幾點：

- 匯入工作支援特定的 SOP 類別和傳輸語法。如需詳細資訊，請參閱 迪克姆支持。
- 在匯入期間，長度限制適用於特定的 DICOM 元素。若要確保匯入工作成功，請確認您的醫療影像資料沒有超過長度限制。如需詳細資訊，請參閱 DICOM 元素限制。
- 匯入工作開始時會執行像素資料驗證檢查。如需詳細資訊，請參閱 像素數據驗證。
- 有與 HealthImaging 匯入動作相關聯的端點、配額和節流限制。如需詳細資訊，請參閱 端點和配額 及 節流限制。
- 對於每個匯入工作，處理結果都會儲存在該`outputS3Uri`位置。處理結果會組織為`job-output-manifest.json`檔案`SUCCESS`和`FAILURE`資料夾。

> **Note**
>
> 單一匯入工作最多可包含 10,000 個巢狀資料夾。

- 該`job-output-manifest.json`文件包含有關處理數據的`jobSummary`輸出和其他詳細信息。下面的例子顯示了一個`job-output-manifest.json`文件的輸出。

```
{
    "jobSummary": {
        "jobId": "09876543210987654321098765432109",
        "datastoreId": "12345678901234567890123456789012",
```

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

- 資SUCCESS料夾會保存包含所有成功匯入之影像success.ndjson檔案結果的檔案。下面的例子顯示了一個success.ndjson文件的輸出。

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

- 該FAILURE資料夾包含未成功匯入之所有影像failure.ndjson檔案結果的檔案。下面的例子顯示了一個failure.ndjson文件的輸出。

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

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

- 匯入工作會保留在工作清單中 90 天，然後封存。

開始匯入工作

您可以使用此`StartDICOMImportJob`動作開始像素資料驗證檢查，並將大量資料匯入 AWS HealthImaging 資料存放區。匯入任務會匯入位於參數指定之 Amazon S3 輸入儲存貯體中的 DICOM P10 檔案。`inputS3Uri`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**
>
> 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，請在啟動匯入工作時提供`inputOwnerAccountId`參數。如需詳細資訊，請參閱 跨帳戶匯入 AWS HealthImaging。
> 在匯入期間，長度限制會套用至特定的 DICOM 元素。如需詳細資訊，請參閱 DICOM 元素限制。

下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請`StartDICOMImportJob`參閱 AWS HealthImaging API 參考中的。

啟動匯入工作

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。
3. 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 (選填)

- S3 中的匯入來源位置
- 來源時段擁有者的帳戶 ID (選擇性)
- 加密金鑰 (選擇性)
- 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 和軟體開發套件

C++

適用於 C++ 的 SDK

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

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

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

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

    return startDICOMImportJobOutcome.IsSuccess();
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK for C++ API 參考ImportJob中的開始數據。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

若要啟動讀入工作

下列start-dicom-import-job程式碼範例會啟動 dicom 匯入工作。

```
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，請參閱開發AWS HealthImaging 人員指南中的開始匯入工作。

- 如需 API 的詳細資訊，請參閱AWS CLI 命令參考ImportJob中的 StartICOM。

Java

適用於 Java 2.x 的 SDK

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

        try {
            StartDicomImportJobRequest startDicomImportJobRequest =
 StartDicomImportJobRequest.builder()
                    .jobName(jobName)
                    .datastoreId(datastoreId)
                    .dataAccessRoleArn(dataAccessRoleArn)
                    .inputS3Uri(inputS3Uri)
```

```
                    .outputS3Uri(outputS3Uri)
                    .build();
            StartDicomImportJobResponse response =
 medicalImagingClient.startDICOMImportJob(startDicomImportJobRequest);
            return response.jobId();
        } catch (MedicalImagingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return "";
    }
```

- 有關 API 的詳細信息，請參閱 AWS SDK for Java 2.x API 參考ImportJob中的開始數據。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

- 有關 API 的詳細信息，請參閱 AWS SDK for JavaScript API 參考ImportJob中的開始數據。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 有關 API 的詳細信息，請參閱開始AWS使用 SDK ImportJob 中的 Python（博托 3）API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

取得匯入工作屬性

您可以使用此GetDICOMImportJob動作進一步了解 AWS HealthImaging 匯入任務屬性。例如，在開始匯入工作之後，您可以執行GetDICOMImportJob以尋找工作的狀態。一旦jobStatus返回為COMPLETED，您就可以訪問您的圖像集。

> **Note**
>
> jobStatus指的是匯入工作的執行。因此，COMPLETED即使在匯入程序期間發現驗證問題，匯入工作也可以傳回 a jobStatus。如果jobStatus退貨為COMPLETED，我們仍建議您檢閱寫入 Amazon S3 的輸出資訊清單，因為這些資訊清單會提供個別 P10 物件匯入成功或失敗的詳細資訊。

下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請GetDICOMImportJob參閱 AWS HealthImaging API 參考中的。

取得匯入工作屬性

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

資料倉庫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」頁標。
4. 選擇匯入工作。

[匯入工作詳細資訊] 頁面隨即開啟並顯示匯入工作的相關內容。

AWS CLI 和軟體開發套件

C++

適用於 C++ 的 SDK

```
//! 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 的詳細資訊，請參閱 AWS SDK for C++ API 參考資料ImportJob中的 GetDicom。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

若要取得 DICOM 匯入工作的屬性

下列get-dicom-import-job程式碼範例會取得 dicom 匯入工作的屬性。

```
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，請參閱開AWS HealthImaging 發人員指南中的取得匯入工作屬性。

- 如需 API 的詳細資訊，請參閱AWS CLI 命令參考ImportJob中的 GetDicom。

Java

適用於 Java 2.x 的 SDK

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

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

       return null;
   }
```

- 如需 API 的詳細資訊，請參閱 AWS SDK for Java 2.x API 參考資料ImportJob中的 GetDicom。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

  return response;
};
```

- 如需 API 的詳細資訊，請參閱 AWS SDK for JavaScript API 參考資料ImportJob中的 GetDicom。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ImportJob中的 GetDCOM (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

列出匯入工作

您可以使用`ListDICOMImportJobs`動作列出針對特定 HealthImaging 資料倉庫建立的匯入工作。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請`ListDICOMImportJobs`參閱 AWS HealthImaging API 參考中的。

> **Note**
>
> 匯入工作會保留在工作清單中 90 天，然後封存。

列出匯入工作

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   資料倉庫詳細資訊頁面隨即開啟。依預設，會選取 [影像集] 索引標籤。
3. 選擇「匯入」頁籤以列出所有相關聯的匯入工作。

AWS CLI 和軟體開發套件

CLI

AWS CLI

若要列出讀入工作

下列`list-dicom-import-jobs`程式碼範例會列出 dicom 匯入工作。

```
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工作。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ImportJobs中的清單 Dicom。

Java

適用於 Java 2.x 的 SDK

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

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

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

- 如需 API 詳細資訊，請參閱 API 參考資料ImportJobs中的清單AWS SDK for Java 2.x介面。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

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

  return jobSummaries;
};
```

- 如需 API 詳細資訊，請參閱 API 參考資料ImportJobs中的清單AWS SDK for JavaScript介面。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ImportJobs中的列表 DCOM (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

使用 AWS 存取映像集 HealthImaging

在 AWS 中存取醫學影像資料 HealthImaging 通常涉及搜尋具有唯一金鑰的影像集，以及取得相關的中繼資料和影像畫面 (像素資料)。

> **Tip**
>
> 熟悉 AWS 之後 HealthImaging，我們建議您瀏覽以使用我們的檢視專案AWS HealthImaging 範例專案進行實作快速啟動。

下列主題說明什麼是影像集，以及如何使用 AWS Management Console AWS CLI、和 AWS SDK 來搜尋這些影像集，並取得其關聯的屬性、中繼資料和影像框。

主題

瞭解影像集

映像集是做為 AWS 基礎的 AWS 概念 HealthImaging。影像集是在您匯入 DICOM 資料時建立的 HealthImaging，因此在使用服務時需要對它們有很好的了解。

引入影像集的原因如下：

- 透過靈活的 API Support 各種醫學影像工作流程 (臨床和非臨床)。
- 通過僅分組相關數據來最大限度地提高患者
- 鼓勵清理資料，以協助提高不一致的可見度。如需詳細資訊，請參閱 修改影像集。

> **重要**
>
> 在清除 DICOM 數據之前臨床使用可能會導致患者傷害。

下列功能表會進一步詳細描述影像集，並提供範例和圖表，以協助您瞭解影像集的功能和用途。HealthImaging

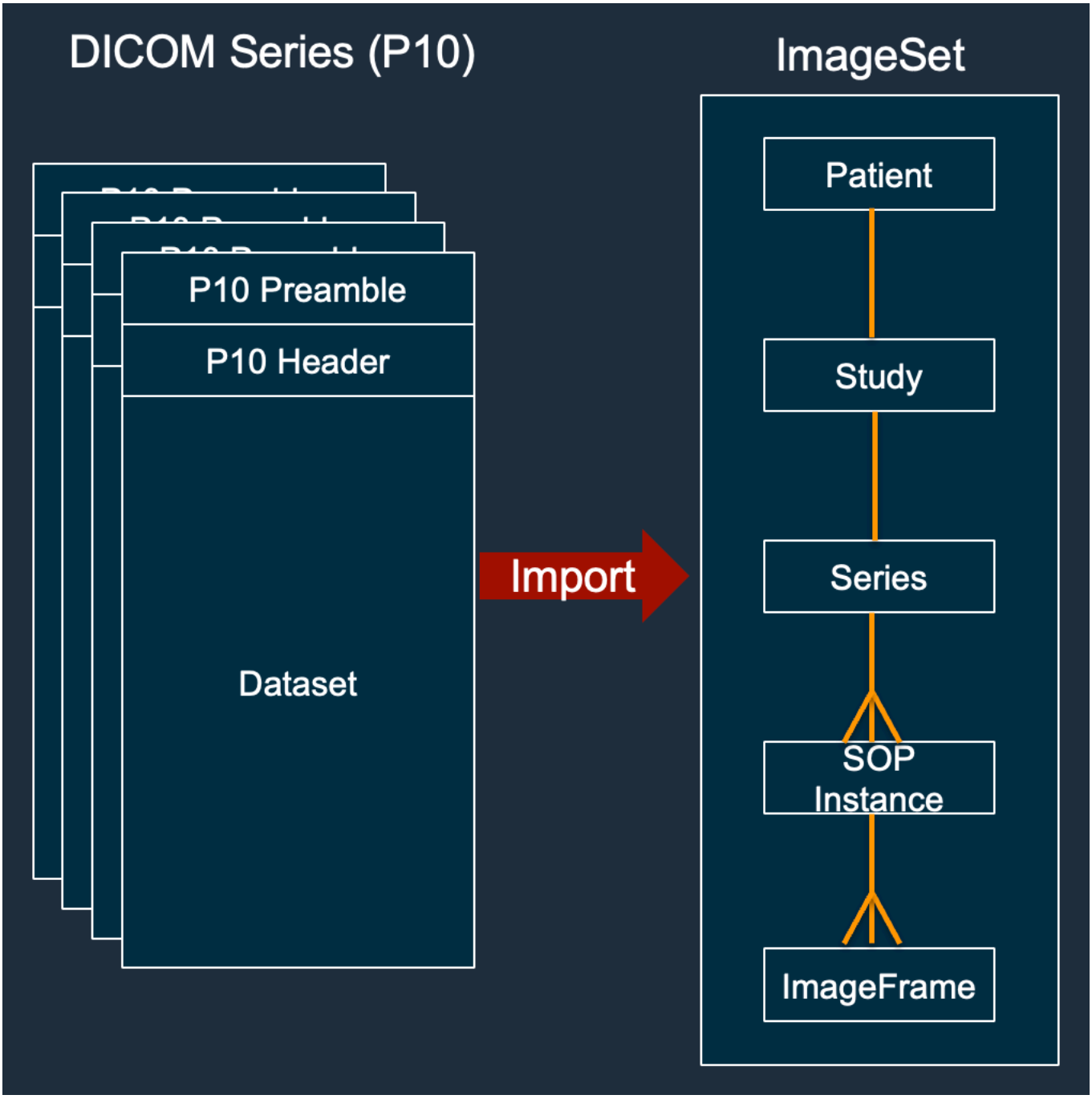
什麼是影像集？

影像集是定義用於最佳化相關醫學影像資料的抽象分組機制的 AWS 概念。將 DICOM P10 影像資料匯入 AWS 資 HealthImaging 料存放區時，它會轉換為由中繼資料和影像畫面 (像素資料) 組成的映像集。

> **Note**
>
> 影像集中繼資料已標準化。換句話說，一組常見的屬性和值會對應至 DICOM 資料元素登錄中列出的「病患」、「研究」和「系列」層級元素。
> 影像畫面 (像素資料) 會以高傳輸量 JPEG 2000 (HTJ2K) 編碼，且必須先解碼才能檢視。

映像集是 AWS 資源，因此會指派 Amazon 資源名稱 (ARN)。它們可以使用多達 50 個鍵值對和授予的基於角色的訪問控制（RBAC）和基於屬性的訪問控制（ABAC）通過 IAM 進行標記。此外，影像集已建立版本化，因此會保留所有變更，並可存取先前的版本。

匯入 DICOM P10 資料會產生包含相同 DICOM 系列中之一或多個服務-物件配對 (SOP) 執行個體的 DICOM 中繼資料和影像框架的映像集。



Note

匯入工作：

- 永遠建立新的影像集，絕不更新現有的影像集。
- 請勿刪除重複 SOP 執行個體儲存體，因為相同 SOP 執行個體的每次匯入都會使用額外的儲存體

- 可以為單一 DICOM 系列建立多個影像集。例如，當存在規範化中繼資料屬性 (例如不相PatientName符) 的變體時。

影像設定中繼資料是什麼樣子？

您可以使用GetImageSetMetadata動作擷取影像集中繼資料。傳回的中繼資料會以壓縮gzip，因此您必須先將其解壓縮，才能檢視。如需詳細資訊，請參閱 取得影像集中繼資料。

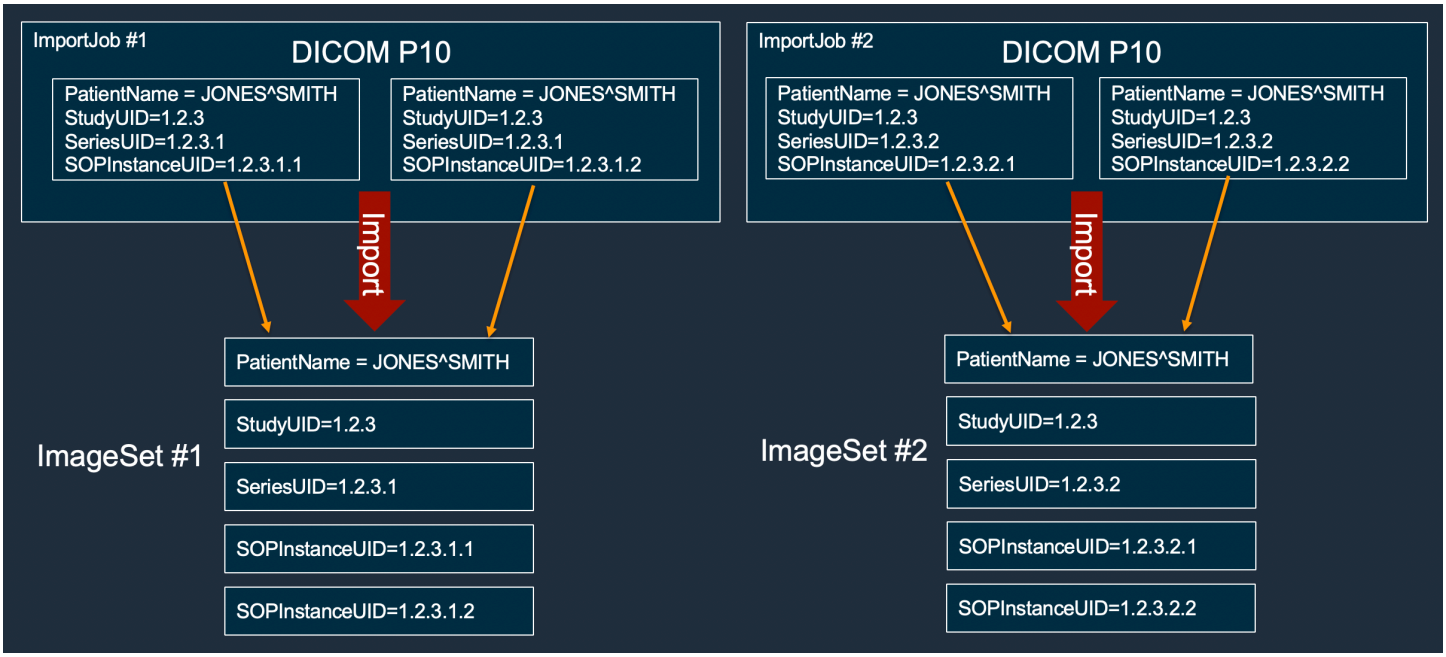
下列範例會顯示 JSON 格式的影像集中繼資料的結構。

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

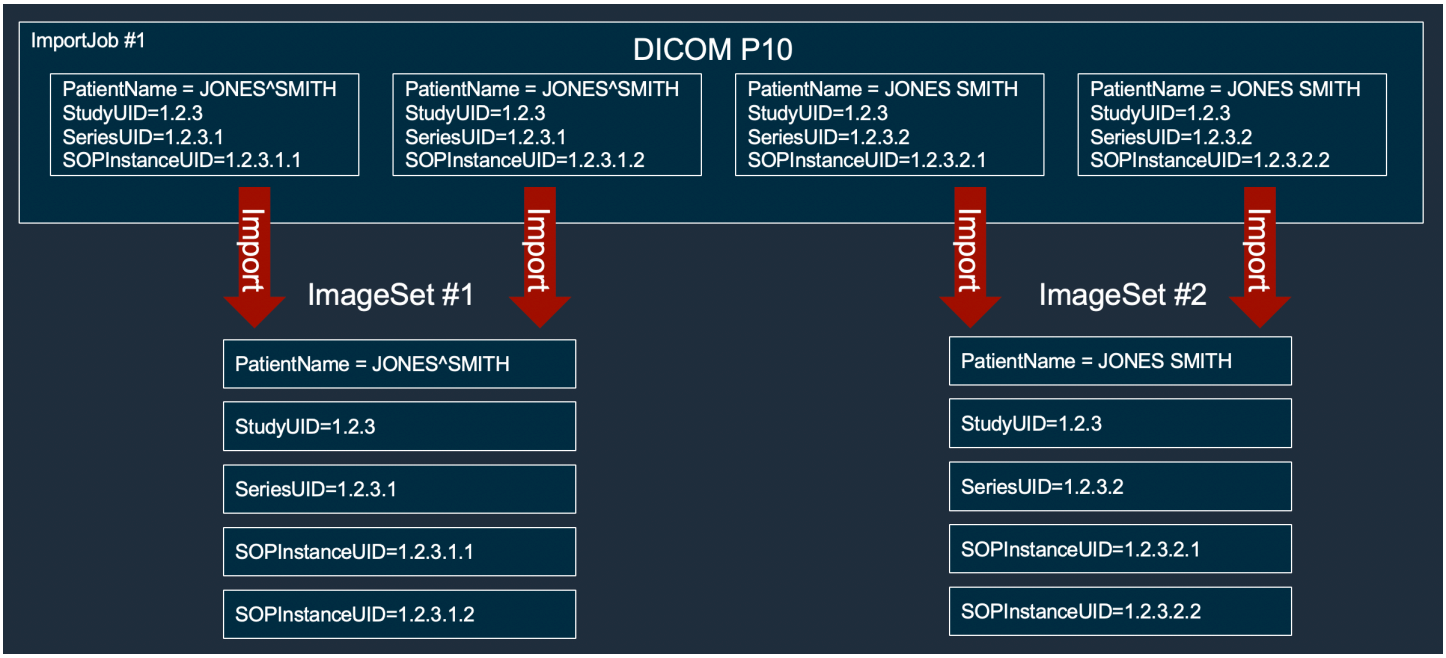
影像集建立範例：多個匯入工作

下列範例顯示多個匯入工作如何一律建立新的影像集，而不是新增至現有的影像集。



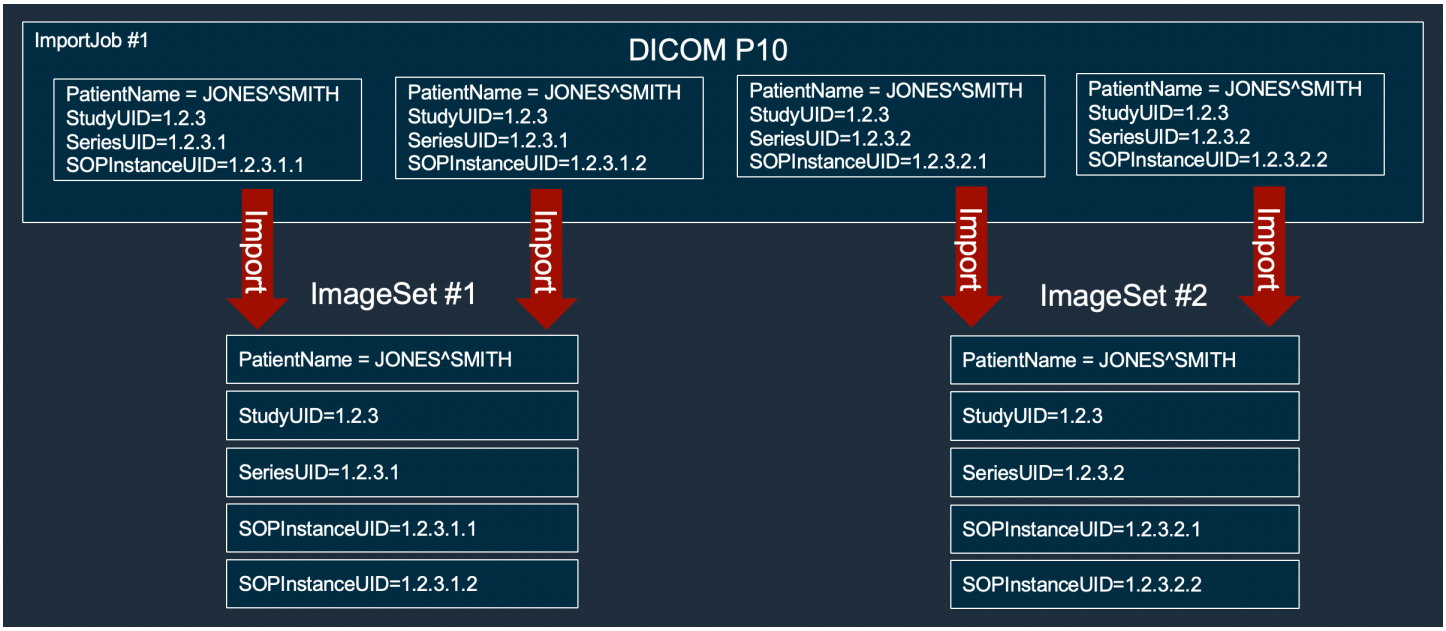
影像集建立範例：具有兩個變體的單一匯入工作

下列範例顯示建立兩個映像集的單一匯入工作，因為執行個體 1 和 2 的病患名稱與執行個體 3 和 4 不同。



影像集建立範例：具有最佳化的單一匯入工作

下列範例顯示單一匯入工作，即使病患名稱相符，建立兩個影像集以改善輸送量。



搜尋影像集

您可以使用此SearchImageSets動作針對ACTIVE HealthImaging 資料倉庫中的所有影像集執行搜尋查詢。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請SearchImageSets參閱 AWS HealthImaging API 參考中的。

> **Note**
>
> 搜尋影像集時，請記住以下幾點。
>
> - 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)。
> - 依預設，SearchImageSets使用updatedAt欄位以從最新到最舊的遞減順序進行排序。
> - 如果您使用客戶擁有的 AWS KMS 金鑰建立資料存放區，則必須在與映像集互動之前更新 AWS KMS 金鑰政策。如需詳細資訊，請參閱建立客戶管理的金鑰。

搜尋影像集

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

> **Note**
>
> 下列程序展示如何使用`Series Instance UID`和`Updated at`屬性篩選器搜尋影像集。

Series Instance UID

使用`Series Instance UID`屬性篩選器搜尋影像集

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   預設情況下，「資料倉庫詳細資訊」頁面會開啟，並選取「影像集」索引標
3. 選擇屬性篩選功能表並選取`Series Instance UID`。
4. 在「輸入要搜尋的值」欄位中，輸入 (貼上) 感興趣的「序列執行環境 UID」。

   > **Note**
   >
   > 序列執行個體 UID 值必須與 DICOM 唯一識別碼 (UID) 登錄中列出的值相同。請注意，需求包括一系列數字，它們之間至少包含一個句點。系列執行個體 UID 的開始或結尾不允許使用期間。不允許使用字母和空格，因此在複製和粘貼 UID 時請小心。
5. 選擇「日期範圍」功能表，選取「序列執行環境 UID」的日期範圍，然後選擇「套用」。
6. 選擇 Search (搜尋)。

   依預設，會以最新順序傳回落在所選日期範圍內的序列執行個體 UID。

Updated at

使用`Updated at`屬性篩選器搜尋影像集

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   預設情況下，「資料倉庫詳細資訊」頁面會開啟，並選取「影像集」索引標
3. 選擇屬性篩選功能表，然後選擇`Updated at`。

4. 選擇「日期範圍」選單，選取影像集日期範圍，然後選擇「套用」。
5. 選擇 Search (搜尋)。

   依預設，會以最新順序傳回落在所選日期範圍內的影像集。

AWS CLI 和軟體開發套件

C++

適用於 C++ 的 SDK

用於搜索圖像集的實用程序功能。

```
//! 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: 等於運算子。

```
        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::Oper

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

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

 searchCriteriaEqualsPatientID,

 imageIDsForPatientID,
                                                                clientConfig);
        if (result) {
            std::cout << imageIDsForPatientID.size() << " image sets found for
 the patient with ID '"
```

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

使用案例 #2: 使用 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

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

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

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

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

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

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

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

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

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

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

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

useCase3StartDate.SetCreatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase3EndDate;

useCase3EndDate.SetCreatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));

        Aws::MedicalImaging::Model::SearchFilter useCase3SearchFilter;
        useCase3SearchFilter.SetValues({useCase3StartDate, useCase3EndDate});

useCase3SearchFilter.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchCriteria useCase3SearchCriteria;
        useCase3SearchCriteria.SetFilters({useCase3SearchFilter});

        Aws::Vector<Aws::String> usesCase3Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase3SearchCriteria,
                                                          usesCase3Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase3Results.size() << " image sets found for
created between 2023/11/30 and present."
                      <<  std::endl;
            for (auto &imageSetResult : usesCase3Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 ASC 順序排序回應。

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

useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase4EndDate;

useCase4EndDate.SetUpdatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterBetween;
        useCase4SearchFilterBetween.SetValues({useCase4StartDate,
useCase4EndDate});

useCase4SearchFilterBetween.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchByAttributeValue seriesInstanceUID;
        seriesInstanceUID.SetDICOMSeriesInstanceUID(dicomSeriesInstanceUID);

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterEqual;
        useCase4SearchFilterEqual.SetValues({seriesInstanceUID});

useCase4SearchFilterEqual.SetOperator(Aws::MedicalImaging::Model::Operator::EQUAL);

        Aws::MedicalImaging::Model::SearchCriteria useCase4SearchCriteria;
        useCase4SearchCriteria.SetFilters({useCase4SearchFilterBetween,
useCase4SearchFilterEqual});

        Aws::MedicalImaging::Model::Sort useCase4Sort;

useCase4Sort.SetSortField(Aws::MedicalImaging::Model::SortField::updatedAt);
        useCase4Sort.SetSortOrder(Aws::MedicalImaging::Model::SortOrder::ASC);

        useCase4SearchCriteria.SetSort(useCase4Sort);

        Aws::Vector<Aws::String> usesCase4Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase4SearchCriteria,
                                                          usesCase4Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase4Results.size() << " image sets found for EQUAL
operator "
            << "on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort
response\n"
```

```
            <<  "in ASC order on updatedAt field." <<  std::endl;
            for (auto &imageSetResult : usesCase4Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
 std::endl;
            }
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考SearchImageSets中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

範例 1：使用 EQUAL 運算子搜尋影像集

下列search-image-sets程式碼範例會使用 EQUAL 運算子，根據特定值來搜尋影像集。

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

search-criteria.json 的內容

```
{
    "filters": [{
        "values": [{"DICOMPatientId" : "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": "1.2.840.99999999.84710745.943275268089",
             "DICOMPatientBirthDate": "19201120",
             "DICOMStudyDescription": "UNKNOWN",
             "DICOMPatientId": "SUBJECT08701",
             "DICOMPatientName": "Melissa844 Huel628",
             "DICOMNumberOfStudyRelatedInstances": 1,
             "DICOMStudyTime": "140728",
             "DICOMNumberOfStudyRelatedSeries": 1
            },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

範例 2：若要使用 DICOM StudyDate 和 DICOM 使用「之間」運算子來搜尋影像集 StudyTime

下列search-image-sets程式碼範例會搜尋包含在 1990 年 1 月 1 日 (上午 12 時) 至 2023 年 1 月 1 日 (上午 12 時) 之間產生之 DICOM 研究的影像集。

注意：DICOM StudyTime 是可選的。如果不存在，12:00 AM（當天的開始）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": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

範例 3：若要使用 createdAt 使用 BETWEEN 運算子搜尋影像集 (先前已保留時間研究)

下列`search-image-sets`程式碼範例會搜尋 DICOM 研究在 UTC 時區時間範圍 HealthImaging 之間持續存在的影像集。

注意：以範例格式提供「createdAt 資訊」(「一九八五年四月十二時 20:50.52 Z」)。

```
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": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

範例 4：若要在 DICOM SeriesInstance UID 上使用 EQUAL 運算子搜尋影像集，並在更新時以 ASC 順序排序回應。

下列search-image-sets程式碼範例會在 DICOM SeriesInstance UID 上使用 EQUAL 運算子搜尋影像集，並在 updatedAt 上以 ASC 順序排序回應。

注意：以範例格式提供 updatedAt。(「1985 年 4 月 12 日 23:20：50.52 Z」)。

```
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": "1.2.840.99999999.84710745.943275268089"
        }],
        "operator": "EQUAL"
    }],
    "sort": {
        "sortField": "updatedAt",
        "sortOrder": "ASC"
    }
}
```

輸出：

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
```

```
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

如需詳細資訊，請參閱AWS HealthImaging 開發人員指南中的搜尋影像集。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考SearchImageSets中的。

Java

適用於 Java 2.x 的 SDK

用於搜索圖像集的實用程序功能。

```
    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: 等於運算子。

```
        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 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

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

 .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
                                        .dicomStudyDate("19990101")
                                        .dicomStudyTime("000000.000")
```

```
                                        .build())
                                .build(),
                        SearchByAttributeValue.builder()

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

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

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

```
        searchFilters = Collections.singletonList(SearchFilter.builder()
                .operator(Operator.BETWEEN)
                .values(SearchByAttributeValue.builder()

.createdAt(Instant.parse("1985-04-12T23:20:50.52Z"))
                                .build(),
                        SearchByAttributeValue.builder()
                                .createdAt(Instant.now())
                                .build())
                .build());

        searchCriteria = SearchCriteria.builder()
```

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

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 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SearchImageSets中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

用於搜索圖像集的實用程序功能。

```
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: 等於運算子。

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

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

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

使用案例 #2: 使用 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

```
    const datastoreId = "12345678901234567890123456789012";

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

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

```
const datastoreId = "1234567890123456789012345678901 2";

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

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

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 ASC 順序排序回應。

```
const datastoreId = "1234567890123456789012345678901 2";

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

```
                },
                {
                    values: [
                        {DICOMSeriesInstanceUID:
 "1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
                    ],
                    operator: "EQUAL",
                },
            ],
            sort: {
                sortOrder: "ASC",
                sortField: "updatedAt",
            }
        };

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考SearchImageSets中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

用於搜索圖像集的實用程序功能。

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


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.
```

```
        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.health_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries
```

使用案例 #1: 等於運算子。

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

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

使用案例 #2: 使用 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

```
        search_filter = {
```

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

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

```
        search_filter = {
            "filters": [
                {
                    "values": [
                        {
                            "createdAt": datetime.datetime(
                                2021, 8, 4, 14, 49, 54, 429000
                            )
                        },
                        {
                            "createdAt": datetime.datetime.now()
                            + datetime.timedelta(days=1)
                        },
```

```
                    ],
                    "operator": "BETWEEN",
                }
            ]
        }

        recent_image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            f"Image sets found with with BETWEEN operator using createdAt
\n{recent_image_sets}"
        )
```

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 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}")
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件SearchImageSets中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

取得影像集屬性

您可以使用GetImageSet動作來傳回中設定之指定影像的屬性 HealthImaging。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請GetImageSet參閱 AWS HealthImaging API 參考中的。

> **Note**
>
> 根據預設，AWS 會 HealthImaging 傳回映像集最新版本的屬性。若要檢視舊版影像集的屬性，請提供您versionId的要求。

取得影像集屬性

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   預設情況下，「資料倉庫詳細資訊」頁面會開啟，並選取「影像集」索引標
3. 選擇影像集。

   [影像集詳細資料] 頁面隨即開啟，並顯示影像集屬性。

AWS CLI 和軟體開發套件

CLI

AWS CLI

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，請參閱開AWS HealthImaging 發人員指南中的取得影像集屬性。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetImageSet中的。

Java

適用於 Java 2.x 的 SDK

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

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

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

```
  // }

  return response;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetImageSet中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

取得影像集中繼資料

您可以使用`GetImageSetMetadata`動作擷取中指定影像集的中繼資料 HealthImaging。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請`GetImageSetMetadata`參閱 AWS HealthImaging API 參考中的。

> **Note**
>
> 根據預設，會 HealthImaging 傳回影像集最新版本的中繼資料屬性。若要檢視舊版影像集的中繼資料，請提供您`versionId`的要求。
>
> 影像集中繼資料會以 JSON 物件壓縮`gzip`並傳回。因此，您必須先解壓縮 JSON 物件，才能檢視標準化的中繼資料。如需詳細資訊，請參閱 元数据规范化。

取得影像集中繼資料

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   預設情況下，「資料倉庫詳細資訊」頁面會開啟，並選取「影像集」索引標
3. 選擇影像集。

   影像集詳細資料頁面隨即開啟，影像集中繼資料會顯示在 [影像集中繼資料檢視器] 區段下。

AWS CLI 和軟體開發套件

C++

適用於 C++ 的 SDK

獲取圖像集元數據的實用程序功能。

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

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

    return outcome.IsSuccess();
}
```

獲取沒有版本的圖像集元數據。

```
        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;
        }
```

獲取具有版本的圖像集元數據。

```
        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 AWS SDK for C++ API 參考GetImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

範例 1：若要取得沒有版本的影像集中繼資料

下列get-image-set-metadata程式碼範例會取得影像集的中繼資料，而不指定版本。

注意：outfile是必需的參數

```
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，並儲存在研究中繼資料 .json.gz 檔案中。若要檢視傳回的 JSON 物件的內容，您必須先將其解壓縮。

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影像集的中繼資料。

注意：outfile是必需的參數

```
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，並儲存在研究中繼資料 .json.gz 檔案中。若要檢視傳回的 JSON 物件的內容，您必須先將其解壓縮。

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影像集中繼資料。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetImageSetMetadata中的。

Java

適用於 Java 2.x 的 SDK

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

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

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


medicalImagingClient.getImageSetMetadata(getImageSetMetadataRequestBuilder.build(),
```

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

獲取圖像集元數據的實用程序功能。

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

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

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

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

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

  return response;
};
```

獲取沒有版本的圖像集元數據。

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

獲取具有版本的圖像集元數據。

```
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 AWS SDK for JavaScript API 參考GetImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

獲取圖像集元數據的實用程序功能。

```
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
```

獲取沒有版本的圖像集元數據。

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

獲取具有版本的圖像集元數據。

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

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetImageSetMetadata中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

取得影像集像素資料

圖像幀是存在於圖像集中的像素數據，用於組成 2D 醫學圖像。您可以使用此GetImageFrame動作擷取中指定影像集的 HTJ2K 編碼影像影格。 HealthImaging下列功能表提供 AWS CLI 和 AWS SDK 的程式碼範例。如需詳細資訊，請GetImageFrame參閱 AWS HealthImaging API 參考中的。

> **Note**
>
> 在匯入期間，AWS 會以 HTJ2K 無失真格式對所有映像畫面進行編 HealthImaging 碼，因此必須先將它們解碼，才能在影像檢視器中檢視。如需詳細資訊，請參閱 HTJ2K 解碼程式庫。

若要取得影像框架

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

> **Note**
>
> 影像框必須以程式設計方式存取和解碼，因為中不提供影像檢視器。 AWS Management Console
> 若要取得有關解碼和檢視影像框的更多資訊，請參閱HTJ2K 解碼程式庫。

AWS CLI 和軟體開發套件

C++

適用於 C++ 的 SDK

```
//! 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 AWS SDK for C++ API 參考GetImageFrame中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

取得影像集像素資料

下列get-image-frame程式碼範例會取得影像框。

```
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框的詳細資訊，請參閱 HTJ2K 解碼程式庫。

如需詳細資訊，請參閱開AWS HealthImaging 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GetImageFrame中的。

Java

適用於 Java 2.x 的 SDK

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

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

.imageFrameInformation(ImageFrameInformation.builder()

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

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetImageFrame中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考GetImageFrame中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetImageFrame中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

取得 DICOM 執行個體

> **Note**
>
> 該 HealthImaging GetDICOMInstance API 內置符合 DicomWeb 檢索（WADO-RS）標準，用於基於 Web 的醫療影像。由GetDICOMInstance於是 DicomWeb 服務的表示形式，因此它不是透過 AWS CLI 和 AWS SDK 提供的。

您可GetDICOMInstance以透過指定與 HealthImaging 資源相關聯的「系列」、「研究」和「實例 UID」，從資料存放區擷取 DICOM 實例。您可以提供影像集 ID 作為查詢參數，來指定要從中擷取執行處理資源的影像集。此外，您還可以選擇傳輸語法來壓縮 DICOM 數據，並支持未壓縮（ELE）或高通量 JPEG 2000（HTJ2K）。透過GetDICOMInstance，您可以與使用 DICOM 第 10 部分二進位檔案的 HealthImaging系統互操作，同時利用雲端原生介面。

若要取得 DICOM 實體 (**.dcm**檔案)

1. 收集 HealthImaging datastoreId和imageSetId參數值。
2. 將GetImageSetMetadata動作與datastoreId和imageSetId參數值搭配使用，可擷取studyInstanceUID、seriesInstanceUID和的關聯中繼資料值sopInstanceUID。如需詳細資訊，請參閱 取得影像集中繼資料。
3. 使用datastoreId、studyInstanceUID、、和的值建構要求的 URL imageSetId。seriesInstanceUID sopInstanceUID網址的格式如下：

   ```
   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. 準備並發送您的請求。 GetDICOMInstance使用帶有簽AWS 名版本 4 簽名協議的 HTTP GET 請求。下列程式碼範例會使用curl命令列工具從 HealthImaging中取得 DICOM 實體 (.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 是可選的，如果未包含，則默認為「顯式 VR 小端序」。支援的傳輸語法包括：
>
> - 顯式 VR 小端序（ELE）-`1.2.840.10008.1.2.1`（默認）
> - 採用 RPCL 選項影像壓縮功能的高輸送量 JPEG 2000 (僅限不失真)-`1.2.840.10008.1.2.4.202`
>
> 如需更多詳細資訊，請參閱 適用於 AWS 的 HTJ2K 解碼程式庫 HealthImaging。

使用 AWS 修改映像集 HealthImaging

DICOM 匯入工作通常會要求您修改映像集，原因如下：

- 病人安全
- 資料一致性
- 降低儲存成本

HealthImaging 提供數個 API 來簡化影像集修改程序。下列主題說明如何使用AWS CLI和 AWS SDK 修改影像集。

主題

列出圖像集版本

您可以使用`ListImageSetVersions`動作列出中影像集的版本歷程記錄 HealthImaging。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請`ListImageSetVersions`參閱 AWS HealthImaging API 參考中的。

> **Note**
>
> AWS 會 HealthImaging 記錄對映像集所做的每項變更。更新影像集中繼資料會在影像集歷程記錄中建立新版本。如需詳細資訊，請參閱 更新影像集中繼資料。

列出影像集的版本

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。

2. 選擇資料倉庫。

   預設情況下，「資料倉庫詳細資訊」頁面會開啟，並選取「影像集」索引標

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 [影像集詳細資料] 區段下。

AWS CLI 和軟體開發套件

CLI

AWS CLI

列出影像集版本的步驟

下列`list-image-set-versions`程式碼範例會列出影像集的版本歷程記錄。

```
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ListImageSetVersions中的。

Java

適用於 Java 2.x 的 SDK

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

            ListImageSetVersionsIterable responses = medicalImagingClient
                    .listImageSetVersionsPaginator(getImageSetRequest);
            List<ImageSetProperties> imageSetProperties = new ArrayList<>();
```

```
            responses.stream().forEach(response ->
 imageSetProperties.addAll(response.imageSetPropertiesList()));

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

        return null;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListImageSetVersions中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考ListImageSetVersions中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListImageSetVersions中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

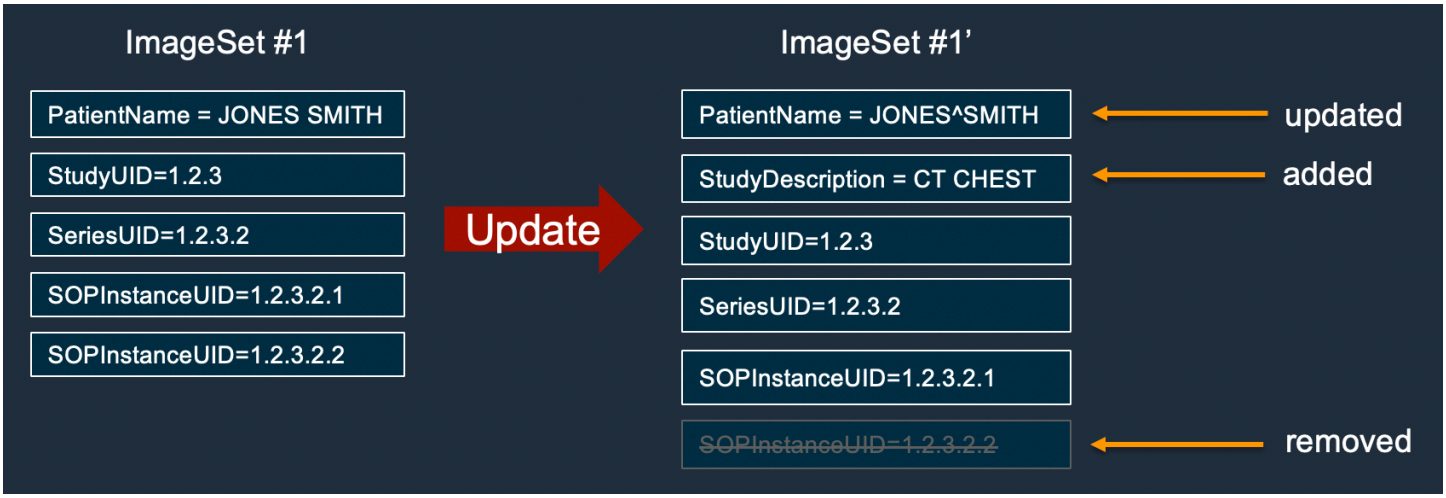
更新影像集中繼資料

您可以使用UpdateImageSetMetadata動作更新 AWS 中的映像集中繼資料 HealthImaging。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 標準化元素的表現形式。您也可以使用此UpdateImageSetMetadata動作移除系列和 SOP 執行個體，以使映像集與外部系統保持同步，並將影像集中繼資料去除識別化。如需詳細資訊，請UpdateImageSetMetadata參閱 AWS HealthImaging API 參考中的。

瞭解影像集中繼資料更新

> **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，請記住以下幾點：
>
> - 更新影像集中繼資料會在影像集歷程記錄中建立新版本。如需詳細資訊，請參閱 列出圖像集版本。
> - 更新影像集中繼資料是一項非同步程序。因此imageSetState，imageSetWorkflowStatus回應元素可用來提供鎖定影像集的個別狀態和狀態。您無法對鎖定的影像集執行其他寫入作業。
> - DICOM 元素條件約束會套用至中繼資料更新。如需詳細資訊，請參閱 DICOM 中繼資料限制。
> - 如果影像集中繼資料更新動作不成功，請呼叫並檢閱message回應元素。

下圖表示正在更新中的圖像集元數據 HealthImaging。



更新影像集詮釋資料

根據您對 AWS 的存取偏好選擇索引標籤 HealthImaging。

AWS CLI 和軟體開發套件

CLI

AWS CLI

在影像集中繼資料中插入或更新屬性的步驟

下列update-image-set-metadata程式碼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 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的內容

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

注意：updatableAttributes是一個以 Base64 編碼的 JSON 字符串。以下是未編碼的 JSON 字符串。

{」SchemaVersion「: 1.1," 病人」: {"迪克姆」: {」PatientName「:" MX^MX "}}

輸出：

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

從影像集中繼資料移除屬性的步驟

下列update-image-set-metadata程式碼範例會從影像集中繼資料移除屬性。

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

metadata-updates.json 的內容

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

注意：removableAttributes是一個以 Base64 編碼的 JSON 字符串。以下是未編碼的 JSON 字符串。鍵和值必須與要刪除的屬性匹配。

{」SchemaVersion「：1.1，「研究」：{「迪克姆」：{」StudyDescription「：「胸部」}}

輸出：

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

若要從影像集中繼資料移除執行個體

下列update-image-set-metadata程式碼範例會從影像集中繼資料移除執行個體。

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

metadata-updates.json 的內容

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

注意：removableAttributes是一個以 Base64 編碼的 JSON 字符串。以下是未編碼的 JSON 字符串。

{"1.1.1.1.1.1.12345.1234567890123.12345678901234.1": {" 執行個體」: {"執行個體」: {"1.1.1.1.1.1.1.12345.123456789034.1": {}}}

輸出：

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

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

如需詳細資訊，請參閱AWS HealthImaging 開發人員指南中的更新影像集中繼資料。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考UpdateImageSetMetadata中的。

Java

適用於 Java 2.x 的 SDK

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

           UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

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);
```

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);
```

使用案例 #3: 移除執行個體。

```
            final String removeInstance = """
                    {
                      "SchemaVersion": 1.1,
                      "Study": {
                        "Series": {
                          "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1":
{
                            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
                          }
                        }
                      }
                    }
                    """;
            MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                    .dicomUpdates(DICOMUpdates.builder()
                            .removableAttributes(SdkBytes.fromByteBuffer(
                                    ByteBuffer.wrap(removeInstance
                                            .getBytes(StandardCharsets.UTF_8))))
                            .build())
                    .build();

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考UpdateImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                              imageSetId = "xxxxxxxxxx",
                                              latestVersionId = "1",
                                              updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
```

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

使用案例 #1: 插入或更新屬性。

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

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

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

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);
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考UpdateImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

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

下面的代碼實例化對 MedicalImagingWrapper 象。

```
    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
            )
```

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
            )
```

使用案例 #3: 移除執行個體。

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

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                                "Instances": {

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

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件UpdateImageSetMetadata中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

複製影像集

您可以使用CopyImageSet動作複製中的影像集 HealthImaging。您可以使用此非同步程序，將影像集的內容複製到新的或現有的影像集中。您可以複製到新影像中以分割影像集，也可以建立單獨的複本。您也可以複製到現有的影像集，將兩個影像集合併在一起。如需詳細資訊，請CopyImageSet參閱 AWS HealthImaging API 參考中的。

了解影像集副本

> **Note**
>
> 複製影像集時，請記住以下幾點：
>
> - 複製影像集會在影像集歷程記錄中建立新版本。如需詳細資訊，請參閱 列出圖像集版本。
> - 複製影像集是一種非同步程序。因此，您可以使用 state (imageSetState) 和 status (imageSetWorkflowStatus) 回應元素，讓您知道鎖定的影像集上發生了什麼作業。無法對鎖定的影像集執行其他寫入作業。
> - CopyImageSet需要唯一的 SOP 執行個體 UID 才能成功。因此，您必須從不需要的映像集中移除來挑選正確的 SOP 執行個體。
> - 如果影像集複製動作不成功，請呼叫GetImageSet並檢閱該message屬性。如需詳細資訊，請參閱 取得影像集屬性。
> - 真實世界的 DICOM 匯入可能會導致每個 DICOM 系列產生多個影像集。使用CopyImageSet動作時，請考慮以下幾點：
>   - 將執行個體從一個映像集複製到另一個
>   - 複製需要兩個影像集具有一致的中繼資料

複製影像集

根據您對 AWS 的存取偏好選擇索引標籤 HealthImaging。

AWS CLI 和軟體開發套件

CLI

AWS CLI

範例 1：複製沒有目的地的影像集。

下列copy-image-set程式碼範例會建立沒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" } }'
```

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：複製具有目標的影像集。

下列copy-image-set程式碼範例會建立具有目標之影像集的複本。

```
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"
}
```

如需詳細資訊，請參閱AWS HealthImaging 開發人員指南中的複製影像集。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CopyImageSet中的。

Java

適用於 Java 2.x 的 SDK

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

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

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

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

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

            CopyImageSetResponse response =
medicalImagingClient.copyImageSet(copyImageSetRequest);

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

        return "";
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考CopyImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

複製影像集的實用程式功能。

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination
 image set.
 * @param {string} destinationVersionId - The optional version ID of the
 destination image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

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

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

複製沒有目的地的影像集。

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

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

複製含有目標的影像集。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考CopyImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

複製影像集的實用程式功能。

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


    def copy_image_set(
        self,
        datastore_id,
        image_set_id,
```

```
        version_id,
        destination_image_set_id=None,
        destination_version_id=None,
    ):
        """
        Copy an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param destination_image_set_id: The ID of the optional destination image
set.
        :param destination_version_id: The ID of the optional destination image
set version.
        :return: The copied image set ID.
        """
        try:
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }
            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
                    "imageSetId": destination_image_set_id,
                    "latestVersionId": destination_version_id,
                }
            copy_results = self.health_imaging_client.copy_image_set(
                datastoreId=datastore_id,
                sourceImageSetId=image_set_id,
                copyImageSetInformation=copy_image_set_information,
            )
        except ClientError as err:
            logger.error(
                "Couldn't copy image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return copy_results["destinationImageSetProperties"]["imageSetId"]
```

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,
            )
```

複製含有目標的影像集。

```
            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,
            )
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件CopyImageSet中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

刪除影像集

您可以使用DeleteImageSet動作刪除中的影像集 HealthImaging。下列功能表提供和 AWS SDK 的程序 AWS Management Console 和程式碼範例。 AWS CLI 如需詳細資訊，請DeleteImageSet參閱 AWS HealthImaging API 參考中的。

刪除影像集

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   預設情況下，「資料倉庫詳細資訊」頁面會開啟，並選取「影像集」索引標
3. 選擇影像集，然後選擇「刪除」。

   刪除影像集」模式隨即開啟。
4. 提供影像集的 ID，然後選擇「刪除影像集」。

AWS CLI 和軟體開發套件

C++

適用於 C++ 的 SDK

```
//! 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 AWS SDK for C++ API 參考DeleteImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

刪除影像集

下列delete-image-set程式碼範例會刪除影像集。

```
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DeleteImageSet中的。

Java

適用於 Java 2.x 的 SDK

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

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DeleteImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

```
  return response;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteImageSet中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

使用 AWS 標記資源 HealthImaging

您可以以標籤的形式將中繼 HealthImaging 資料指派給 AWS 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。

> **重要**
>
> 請勿在標籤中儲存受保護的健康資訊 (PHI)、個人識別資訊 (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 SDK 使用 HealthImaging 標籤作業。如需詳細資訊，請參閱AWS 一般參考 指南中的標記 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

若要在 AWS 中標記資源 HealthImaging，請使用`TagResource`動作。下列程式碼範例說明如何搭配 AWS Management Console AWS CLI、和 AWS SDK 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 AWS 資源。

標記資源 (資料倉庫) 的步驟

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   資料倉庫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」。

   管理標籤」頁面隨即開啟。

5. 選擇 Add new tag (新增標籤)。
6. 輸入「鍵值」與「值」(選擇性)。
7. 選擇儲存變更。

AWS CLI 和軟體開發套件

CLI

AWS CLI

範例 1：為資料倉庫加上標籤

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：標記影像集

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 AWS HealthImaging中的使用標記資源。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考TagResource中的。

Java

適用於 Java 2.x 的 SDK

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

           medicalImagingClient.tagResource(tagResourceRequest);

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考TagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考TagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件TagResource中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

列出資源的標籤

若要列出 AWS 中資源的標籤 HealthImaging，請使用ListTagsForResource動作。下列程式碼範例說明如何搭配 AWS Management Console AWS CLI、和 AWS SDK 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 AWS 資源。

列示資源 (資料倉庫) 標籤的步驟

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   資料倉庫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，會列示所有資料倉庫標籤。

AWS CLI 和軟體開發套件

CLI

AWS CLI

範例 1：列出資料倉庫的資源標籤

下列`list-tags-for-resource`程式碼範例會列出資料倉庫的標籤。

```
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/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 AWS HealthImaging中的使用標記資源。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ListTagsForResource中的。

Java

適用於 Java 2.x 的 SDK

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

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListTagsForResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考ListTagsForResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListTagsForResource中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

取消標記資源

若要在 AWS 中取消標記資源 HealthImaging，請使用UntagResource動作。下列程式碼範例說明如何搭配 AWS Management Console AWS CLI、和 AWS SDK 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 AWS 資源。

取消標記資源 (資料倉庫) 的步驟

根據您對 AWS 的存取偏好選擇功能表 HealthImaging。

AWS 控制台

1. 開啟 HealthImaging 主控台 [資料存放區] 頁面。
2. 選擇資料倉庫。

   資料倉庫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」。

   管理標籤」頁面隨即開啟。
5. 在您要移除的標籤旁選擇「移除」。
6. 選擇儲存變更。

AWS CLI 和軟體開發套件

CLI

AWS CLI

範例 1：取消標籤資料倉庫

下列untag-resource程式碼範例會取消標籤資料存放區。

```
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籤影像集。

```
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 AWS HealthImaging中的使用標記資源。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考UntagResource中的。

Java

適用於 Java 2.x 的 SDK

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

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

            medicalImagingClient.untagResource(untagResourceRequest);

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考UntagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考UntagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件UntagResource中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

HealthImaging 使用 AWS SDK 的程式碼範例

下列程式碼範例顯示如何搭 HealthImaging 配 AWS 軟體開發套件 (SDK) 使用。

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和跨服務範例中查看內容中的動作。

Scenarios (案例) 是向您展示如何呼叫相同服務中的多個函數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，請參閱搭 HealthImaging 配 AWS SDK 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

你好 HealthImaging

下列程式碼範例會示範如何開始使用 HealthImaging。

C++

適用於 C++ 的 SDK

C MakeLists .txt 的 CMake 文件的代碼。

```
# 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 AWS SDK for C++ API 參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

// When no region or credentials are provided, the SDK will use the
```

```
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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

logger = logging.getLogger(__name__)


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

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

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


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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListDatastores中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

程式碼範例

- HealthImaging 使用 AWS SDK 的動作
  - 搭CopyImageSet配 AWS 開發套件或 CLI 使用
  - 搭CreateDatastore配 AWS 開發套件或 CLI 使用
  - 搭DeleteDatastore配 AWS 開發套件或 CLI 使用
  - 搭DeleteImageSet配 AWS 開發套件或 CLI 使用
  - 搭GetDICOMImportJob配 AWS 開發套件或 CLI 使用
  - 搭GetDatastore配 AWS 開發套件或 CLI 使用
  - 搭GetImageFrame配 AWS 開發套件或 CLI 使用

- 搭GetImageSet配 AWS 開發套件或 CLI 使用
- 搭GetImageSetMetadata配 AWS 開發套件或 CLI 使用
- 搭ListDICOMImportJobs配 AWS 開發套件或 CLI 使用
- 搭ListDatastores配 AWS 開發套件或 CLI 使用
- 搭ListImageSetVersions配 AWS 開發套件或 CLI 使用
- 搭ListTagsForResource配 AWS 開發套件或 CLI 使用
- 搭SearchImageSets配 AWS 開發套件或 CLI 使用
- 搭StartDICOMImportJob配 AWS 開發套件或 CLI 使用
- 搭TagResource配 AWS 開發套件或 CLI 使用
- 搭UntagResource配 AWS 開發套件或 CLI 使用
- 搭UpdateImageSetMetadata配 AWS 開發套件或 CLI 使用

- HealthImaging 使用 AWS SDK 的案例
  - 使用 AWS SDK 開始使用影 HealthImaging 像集和影像框
  - 使用 AWS SDK 標記 HealthImaging 資料倉庫
  - 使用 AWS SDK 標記 HealthImaging 影像集

HealthImaging 使用 AWS SDK 的動作

下列程式碼範例示範如何使用 AWS SDK 執 HealthImaging行個別動作。這些摘錄會呼叫 HealthImaging API，是來自必須在內容中執行的大型程式碼摘錄。每個範例都包含一個連結 GitHub，您可以在其中找到設定和執行程式碼的指示。

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

- 搭CopyImageSet配 AWS 開發套件或 CLI 使用
- 搭CreateDatastore配 AWS 開發套件或 CLI 使用
- 搭DeleteDatastore配 AWS 開發套件或 CLI 使用
- 搭DeleteImageSet配 AWS 開發套件或 CLI 使用
- 搭GetDICOMImportJob配 AWS 開發套件或 CLI 使用
- 搭GetDatastore配 AWS 開發套件或 CLI 使用
- 搭GetImageFrame配 AWS 開發套件或 CLI 使用

- 搭GetImageSet配 AWS 開發套件或 CLI 使用
- 搭GetImageSetMetadata配 AWS 開發套件或 CLI 使用
- 搭ListDICOMImportJobs配 AWS 開發套件或 CLI 使用
- 搭ListDatastores配 AWS 開發套件或 CLI 使用
- 搭ListImageSetVersions配 AWS 開發套件或 CLI 使用
- 搭ListTagsForResource配 AWS 開發套件或 CLI 使用
- 搭SearchImageSets配 AWS 開發套件或 CLI 使用
- 搭StartDICOMImportJob配 AWS 開發套件或 CLI 使用
- 搭TagResource配 AWS 開發套件或 CLI 使用
- 搭UntagResource配 AWS 開發套件或 CLI 使用
- 搭UpdateImageSetMetadata配 AWS 開發套件或 CLI 使用

搭**CopyImageSet**配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用CopyImageSet。

CLI

AWS CLI

範例 1：複製沒有目的地的影像集。

下列copy-image-set程式碼範例會建立沒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" } }'
```

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：複製具有目標的影像集。

下列copy-image-set程式碼範例會建立具有目標之影像集的複本。

```
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"
}
```

如需詳細資訊，請參閱AWS HealthImaging 開發人員指南中的複製影像集。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CopyImageSet中的。

Java

適用於 Java 2.x 的 SDK

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

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

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

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

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

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

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

        return "";
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考CopyImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

複製影像集的實用程式功能。

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination
 image set.
 * @param {string} destinationVersionId - The optional version ID of the
 destination image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
```

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

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

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

複製沒有目的地的影像集。

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

複製含有目標的影像集。

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考CopyImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

複製影像集的實用程式功能。

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


    def copy_image_set(
        self,
        datastore_id,
        image_set_id,
        version_id,
        destination_image_set_id=None,
        destination_version_id=None,
    ):
        """
        Copy an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The ID of the image set version.
        :param destination_image_set_id: The ID of the optional destination image
 set.
        :param destination_version_id: The ID of the optional destination image
 set version.
        :return: The copied image set ID.
        """
        try:
            copy_image_set_information = {
                "sourceImageSet": {"latestVersionId": version_id}
            }
            if destination_image_set_id and destination_version_id:
                copy_image_set_information["destinationImageSet"] = {
```

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

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,
            )
```

複製含有目標的影像集。

```
            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,
        )
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件CopyImageSet中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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`CreateDatastore`配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`CreateDatastore`。

Bash

AWS CLI 與 Bash 腳本

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

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

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

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

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

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

  local error_code=${?}

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

  echo "$response"

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CreateDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

建立資料倉庫的步驟

下列create-datastore程式碼範例會建立名稱的資料存放區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倉庫。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CreateDatastore中的。

Java

適用於 Java 2.x 的 SDK

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

       return "";
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考CreateDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考CreateDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件CreateDatastore中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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**DeleteDatastore**配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DeleteDatastore。

Bash

AWS CLI 與 Bash 腳本

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

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

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

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

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

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

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

  local error_code=${?}

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

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

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DeleteDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

刪除資料倉庫的步驟

下列delete-datastore程式碼範例會刪除資料倉庫。

```
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倉庫。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DeleteDatastore中的。

Java

適用於 Java 2.x 的 SDK

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DeleteDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

  return response;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteDatastore中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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DeleteImageSet配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 開始使用影像集和影像框

C++

適用於 C++ 的 SDK

```
//! 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 AWS SDK for C++ API 參考DeleteImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

刪除影像集

下列delete-image-set程式碼範例會刪除影像集。

```
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DeleteImageSet中的。

Java

適用於 Java 2.x 的 SDK

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

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);
```

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考DeleteImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteImageSet中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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GetDICOMImportJob配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 開始使用影像集和影像框

C++

適用於 C++ 的 SDK

```
//! 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 的詳細資訊，請參閱 AWS SDK for C++ API 參考ImportJob中的 GetDicom。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

若要取得 DICOM 匯入工作的屬性

下列get-dicom-import-job程式碼範例會取得 dicom 匯入工作的屬性。

```
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，請參閱開AWS HealthImaging 發人員指南中的取得匯入工作屬性。

- 如需 API 的詳細資訊，請參閱AWS CLI 命令參考ImportJob中的 GetDicom。

Java

適用於 Java 2.x 的 SDK

```
    public static DICOMImportJobProperties getDicomImportJob(MedicalImagingClient
 medicalImagingClient,
            String datastoreId,
```

```
            String jobId) {

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

        return null;
    }
```

- 如需 API 的詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ImportJob中的 GetDicom。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

  return response;
};
```

- 如需 API 的詳細資訊，請參閱 AWS SDK for JavaScript API 參考ImportJob中的 GetDicom。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ImportJob中的 GetDCOM (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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GetDatastore配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetDatastore。

Bash

AWS CLI 與 Bash 腳本

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

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

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

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

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

 local response

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

```
  )
  error_code=${?}

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

  echo "$response"

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

取得資料倉庫性質的步驟

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GetDatastore中的。

Java

適用於 Java 2.x 的 SDK

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetDatastore中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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GetImageFrame配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 開始使用影像集和影像框

C++

適用於 C++ 的 SDK

```
//! 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 AWS SDK for C++ API 參考GetImageFrame中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

取得影像集像素資料

下列get-image-frame程式碼範例會取得影像框。

```
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框的詳細資訊，請參閱 HTJ2K 解碼程式庫。

如需詳細資訊，請參閱開AWS HealthImaging 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GetImageFrame中的。

Java

適用於 Java 2.x 的 SDK

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

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

.imageFrameInformation(ImageFrameInformation.builder()

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

```
 medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetImageFrame中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考GetImageFrame中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetImageFrame中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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GetImageSet配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetImageSet。

CLI

AWS CLI

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，請參閱開AWS HealthImaging 發人員指南中的取得影像集屬性。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetImageSet中的。

Java

適用於 Java 2.x 的 SDK

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

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

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

```
  // }

  return response;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetImageSet中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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GetImageSetMetadata配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 開始使用影像集和影像框

C++

適用於 C++ 的 SDK

獲取圖像集元數據的實用程序功能。

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

    return outcome.IsSuccess();
```

```
}
```

獲取沒有版本的圖像集元數據。

```
        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;
        }
```

獲取具有版本的圖像集元數據。

```
        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 AWS SDK for C++ API 參考GetImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

範例 1：若要取得沒有版本的影像集中繼資料

下列get-image-set-metadata程式碼範例會取得影像集的中繼資料，而不指定版本。

注意：outfile是必需的參數

```
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，並儲存在研究中繼資料 .json.gz 檔案中。若要檢視傳回的 JSON 物件的內容，您必須先將其解壓縮。

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影像集的中繼資料。

注意：outfile是必需的參數

```
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，並儲存在研究中繼資料 .json.gz 檔案中。若要檢視傳回的 JSON 物件的內容，您必須先將其解壓縮。

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影像集中繼資料。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetImageSetMetadata中的。

Java

適用於 Java 2.x 的 SDK

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

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

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


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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

獲取圖像集元數據的實用程序功能。

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

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

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

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

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

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

  return response;
};
```

獲取沒有版本的圖像集元數據。

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

獲取具有版本的圖像集元數據。

```
  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 AWS SDK for JavaScript API 參考GetImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

獲取圖像集元數據的實用程序功能。

```
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
```

獲取沒有版本的圖像集元數據。

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

獲取具有版本的圖像集元數據。

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

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetImageSetMetadata中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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`ListDICOMImportJobs`配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`ListDICOMImportJobs`。

CLI

AWS CLI

若要列出讀入工作

下列`list-dicom-import-jobs`程式碼範例會列出 dicom 匯入工作。

```
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工作。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ImportJobs中的清單 Dicom。

Java

適用於 Java 2.x 的 SDK

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

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

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

- 有關 API 的詳細信息，請參閱 AWS SDK for Java 2.x API 參考ImportJobs中的列表。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

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

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

  return jobSummaries;
};
```

- 有關 API 的詳細信息，請參閱 AWS SDK for JavaScript API 參考ImportJobs中的列表。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

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

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ImportJobs中的列表 DCOM (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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`ListDatastores`配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`ListDatastores`。

Bash

AWS CLI 與 Bash 腳本

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

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

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

  local response
  response=$(aws medical-imaging list-datastores \
    --output text \
```

```
    --query "datastoreSummaries[*][datastoreName, datastoreId, datastoreStatus]")
  error_code=${?}

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

  echo "$response"

  return 0
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

列示資料倉庫的步驟

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ListDatastores中的。

Java

適用於 Java 2.x 的 SDK

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

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

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
```

```
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

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

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListDatastores中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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**ListImageSetVersions**配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`ListImageSetVersions`。

CLI

AWS CLI

列出影像集版本

下列list-image-set-versions程式碼範例會列出影像集的版本歷程記錄。

```
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ListImageSetVersions中的。

Java

適用於 Java 2.x 的 SDK

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

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

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

        return null;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListImageSetVersions中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

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

```
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考ListImageSetVersions中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

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

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListImageSetVersions中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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ListTagsForResource配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 為資料倉庫加標籤
- 標記影像集

CLI

AWS CLI

範例 1：列出資料倉庫的資源標籤

下列list-tags-for-resource程式碼範例會列出資料倉庫的標籤。

```
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/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 AWS HealthImaging中的使用標記資源。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考ListTagsForResource中的。

Java

適用於 Java 2.x 的 SDK

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

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

       return null;
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListTagsForResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考ListTagsForResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListTagsForResource中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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SearchImageSets配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 開始使用影像集和影像框

C++

適用於 C++ 的 SDK

用於搜索圖像集的實用程序功能。

```
//! 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: 等於運算子。

```
        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::Oper

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

```
        };

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

 searchCriteriaEqualsPatientID,

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

使用案例 #2: 使用 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

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

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

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

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

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

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

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

        Aws::MedicalImaging::Model::SearchCriteria useCase2SearchCriteria;
```

```
        useCase2SearchCriteria.SetFilters({useCase2SearchFilter});

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

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

useCase3StartDate.SetCreatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase3EndDate;

useCase3EndDate.SetCreatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));

        Aws::MedicalImaging::Model::SearchFilter useCase3SearchFilter;
        useCase3SearchFilter.SetValues({useCase3StartDate, useCase3EndDate});

useCase3SearchFilter.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchCriteria useCase3SearchCriteria;
        useCase3SearchCriteria.SetFilters({useCase3SearchFilter});

        Aws::Vector<Aws::String> usesCase3Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase3SearchCriteria,
                                                          usesCase3Results,
                                                          clientConfig);
        if (result) {
```

```
            std::cout << usesCase3Results.size() << " image sets found for
created between 2023/11/30 and present."
                      <<  std::endl;
            for (auto &imageSetResult : usesCase3Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 ASC 順序排序回應。

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

useCase4StartDate.SetUpdatedAt(Aws::Utils::DateTime("20231130T000000000Z",Aws::Utils::Da

        Aws::MedicalImaging::Model::SearchByAttributeValue useCase4EndDate;

useCase4EndDate.SetUpdatedAt(Aws::Utils::DateTime(std::chrono::system_clock::now()));

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterBetween;
        useCase4SearchFilterBetween.SetValues({useCase4StartDate,
useCase4EndDate});

useCase4SearchFilterBetween.SetOperator(Aws::MedicalImaging::Model::Operator::BETWEEN);

        Aws::MedicalImaging::Model::SearchByAttributeValue seriesInstanceUID;
        seriesInstanceUID.SetDICOMSeriesInstanceUID(dicomSeriesInstanceUID);

        Aws::MedicalImaging::Model::SearchFilter useCase4SearchFilterEqual;
        useCase4SearchFilterEqual.SetValues({seriesInstanceUID});

useCase4SearchFilterEqual.SetOperator(Aws::MedicalImaging::Model::Operator::EQUAL);

        Aws::MedicalImaging::Model::SearchCriteria useCase4SearchCriteria;
        useCase4SearchCriteria.SetFilters({useCase4SearchFilterBetween,
useCase4SearchFilterEqual});

        Aws::MedicalImaging::Model::Sort useCase4Sort;

useCase4Sort.SetSortField(Aws::MedicalImaging::Model::SortField::updatedAt);
        useCase4Sort.SetSortOrder(Aws::MedicalImaging::Model::SortOrder::ASC);
```

```
        useCase4SearchCriteria.SetSort(useCase4Sort);

        Aws::Vector<Aws::String> usesCase4Results;
        result = AwsDoc::Medical_Imaging::searchImageSets(dataStoreID,
                                                          useCase4SearchCriteria,
                                                          usesCase4Results,
                                                          clientConfig);
        if (result) {
            std::cout << usesCase4Results.size() << " image sets found for EQUAL
operator "
            << "on DICOMSeriesInstanceUID and BETWEEN on updatedAt and sort
response\n"
            <<  "in ASC order on updatedAt field." <<  std::endl;
            for (auto &imageSetResult : usesCase4Results) {
                std::cout << "  Image set with ID '" << imageSetResult <<
std::endl;
            }
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考SearchImageSets中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

範例 1：使用 EQUAL 運算子搜尋影像集

下列`search-image-sets`程式碼範例會使用 EQUAL 運算子，根據特定值來搜尋影像集。

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

`search-criteria.json` 的內容

```
{
    "filters": [{
        "values": [{"DICOMPatientId" : "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": "1.2.840.99999999.84710745.943275268089",
             "DICOMPatientBirthDate": "19201120",
             "DICOMStudyDescription": "UNKNOWN",
             "DICOMPatientId": "SUBJECT08701",
             "DICOMPatientName": "Melissa844 Huel628",
             "DICOMNumberOfStudyRelatedInstances": 1,
             "DICOMStudyTime": "140728",
             "DICOMNumberOfStudyRelatedSeries": 1
            },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

範例 2：若要使用 DICOM StudyDate 和 DICOM 使用「之間」運算子來搜尋影像集 StudyTime

下列`search-image-sets`程式碼範例會搜尋包含在 1990 年 1 月 1 日 (上午 12 時) 至 2023 年 1 月 1 日 (上午 12 時) 之間產生之 DICOM 研究的影像集。

注意：DICOM StudyTime 是選擇性的。如果不存在，12:00 AM（當天的開始）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": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
```

```
    }]
}
```

範例 3：若要使用 createdAt 使用 BETWEEN 運算子搜尋影像集 (先前已保留時間研究)

下列search-image-sets程式碼範例會搜尋 DICOM 研究在 UTC 時區時間範圍 HealthImaging 之間持續存在的影像集。

注意：以範例格式提供「createdAt 資訊」(「一九八五年四月十二時 20:50.52 Z」)。

```
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": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
```

```
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

範例 4：若要在 DICOM SeriesInstance UID 上使用 EQUAL 運算子搜尋影像集，並在更新時以 ASC 順序排序回應

下列search-image-sets程式碼範例會在 DICOM SeriesInstance UID 上使用 EQUAL 運算子搜尋影像集，並在 updatedAt 上以 ASC 順序排序回應。

注意：以範例格式提供 updatedAt。(「1985 年 4 月 12 日 23:20：50.52 Z」)。

```
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": "1.2.840.99999999.84710745.943275268089"
        }],
        "operator": "EQUAL"
    }],
    "sort": {
        "sortField": "updatedAt",
        "sortOrder": "ASC"
    }
}
```

輸出：

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

如需詳細資訊，請參閱AWS HealthImaging 開發人員指南中的搜尋影像集。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考SearchImageSets中的。

Java

適用於 Java 2.x 的 SDK

用於搜索圖像集的實用程序功能。

```
    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: 等於運算子。

```
        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 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

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

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

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

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

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

```
        searchFilters = Collections.singletonList(SearchFilter.builder()
                .operator(Operator.BETWEEN)
```

```
                .values(SearchByAttributeValue.builder()

.createdAt(Instant.parse("1985-04-12T23:20:50.52Z"))
                                .build(),
                        SearchByAttributeValue.builder()
                                .createdAt(Instant.now())
                                .build())
                .build());

        searchCriteria = SearchCriteria.builder()
                .filters(searchFilters)
                .build();
        imageSetsMetadataSummaries =
searchMedicalImagingImageSets(medicalImagingClient,
                datastoreId, searchCriteria);
        if (imageSetsMetadataSummaries != null) {
            System.out.println("The image sets searched with BETWEEN operator
using createdAt are:\n "
                    + imageSetsMetadataSummaries);
            System.out.println();
        }
```

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 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SearchImageSets中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

用於搜索圖像集的實用程序功能。

```
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: 等於運算子。

```
    const datastoreId = "12345678901234567890123456789012";

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

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

使用案例 #2: 使用 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

```
    const datastoreId = "12345678901234567890123456789012";

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

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

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

```
const datastoreId = "1234567890123456789012345678901 2";

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

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

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 ASC 順序排序回應。

```
const datastoreId = "1234567890123456789012345678901 2";
```

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

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

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考SearchImageSets中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

用於搜索圖像集的實用程序功能。

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


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
 [{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
 self.health_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries
```

使用案例 #1: 等於運算子。

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

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

使用案例 #2: 使用 DICOM StudyDate 和 DICOM 之間的運算符。StudyTime

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

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

使用案例 #3: 之間運算符使用 createdAt. 時間研究以前被持續存在。

```
        search_filter = {
            "filters": [
                {
                    "values": [
                        {
```

```
                            "createdAt": datetime.datetime(
                                2021, 8, 4, 14, 49, 54, 429000
                            )
                        },
                        {
                            "createdAt": datetime.datetime.now()
                            + datetime.timedelta(days=1)
                        },
                    ],
                    "operator": "BETWEEN",
                }
            ]
        }

        recent_image_sets = self.search_image_sets(data_store_id, search_filter)
        print(
            f"Image sets found with with BETWEEN operator using createdAt
\n{recent_image_sets}"
        )
```

使用案例 #4: DICOM SeriesInstance UID 上的等於運算子和 updatedAt 間在 updatedAt 新時以 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}")
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件SearchImageSets中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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**StartDICOMImportJob**配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 開始使用影像集和影像框

C++

適用於 C++ 的 SDK

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

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

    if (startDICOMImportJobOutcome.IsSuccess()) {
        importJobId = startDICOMImportJobOutcome.GetResult().GetJobId();
    }
    else {
```

```
        std::cerr << "Failed to start DICOM import job because "
                  << startDICOMImportJobOutcome.GetError().GetMessage() <<
 std::endl;
    }

    return startDICOMImportJobOutcome.IsSuccess();
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK for C++ API 參考ImportJob中的開始數據。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

CLI

AWS CLI

若要啟動讀入工作

下列start-dicom-import-job程式碼範例會啟動 dicom 匯入工作。

```
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，請參閱開發AWS HealthImaging 人員指南中的開始匯入工作。

- 如需 API 的詳細資訊，請參閱AWS CLI 命令參考ImportJob中的 StartICOM。

Java

適用於 Java 2.x 的 SDK

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

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

        return "";
    }
```

- 有關 API 的詳細信息，請參閱 AWS SDK for Java 2.x API 參考ImportJob中的開始數據。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

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

- 有關 API 的詳細信息，請參閱 AWS SDK for JavaScript API 參考ImportJob中的開始數據。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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"]
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 有關 API 的詳細信息，請參閱開始AWS使用 SDK ImportJob 中的 Python（博托 3）API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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TagResource配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 為資料倉庫加標籤
- 標記影像集

CLI

AWS CLI

範例 1：為資料倉庫加上標籤

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：標記影像集

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 AWS HealthImaging中的使用標記資源。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考TagResource中的。

Java

適用於 Java 2.x 的 SDK

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

           medicalImagingClient.tagResource(tagResourceRequest);

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考TagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考TagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件TagResource中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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UntagResource配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用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：

- 為資料倉庫加標籤
- 標記影像集

CLI

AWS CLI

範例 1：取消標籤資料倉庫

下列untag-resource程式碼範例會取消標籤資料存放區。

```
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籤影像集。

```
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 AWS HealthImaging中的使用標記資源。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考UntagResource中的。

Java

適用於 Java 2.x 的 SDK

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

           medicalImagingClient.untagResource(untagResourceRequest);

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考UntagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

```
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 AWS SDK for JavaScript API 參考UntagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件UntagResource中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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UpdateImageSetMetadata配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用UpdateImageSetMetadata。

CLI

AWS CLI

在影像集中繼資料中插入或更新屬性的步驟

下列update-image-set-metadata程式碼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 \
    --update-image-set-metadata-updates file://metadata-updates.json
```

metadata-updates.json 的內容

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

注意：updatableAttributes是一個基 64 編碼的 JSON 字符串。以下是未編碼的 JSON 字符串。

{」SchemaVersion「: 1.1," 病人」: {"迪克姆」: {」PatientName「:" MX^MX "}}

輸出：

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

```
    "updatedAt": 1680042257.908,
    "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
    "imageSetState": "LOCKED",
    "createdAt": 1680027126.436,
    "datastoreId": "12345678901234567890123456789012"
}
```

從影像集中繼資料移除屬性的步驟

下列update-image-set-metadata程式碼範例會從影像集中繼資料移除屬性。

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

metadata-updates.json 的內容

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

注意：removableAttributes是一個基 64 編碼的 JSON 字符串。以下是未編碼的 JSON 字符串。鍵和值必須與要刪除的屬性匹配。

{」SchemaVersion「：1.1，「研究」：{「迪克姆」：{」StudyDescription「：「胸部」}}

輸出：

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

```
}
```

若要從影像集中繼資料移除執行個體

下列update-image-set-metadata程式碼範例會從影像集中繼資料移除執行個體。

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

metadata-updates.json 的內容

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

注意：removableAttributes是一個基 64 編碼的 JSON 字符串。以下是未編碼的 JSON 字符串。

{"1.1.1.1.1.1.12345.1234567890123.12345678901234.1": {" 執行個體」: {"執行個體」: {"1.1.1.1.1.1.1.12345.123456789034.1": {}}}

輸出：

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

如需詳細資訊，請參閱AWS HealthImaging 開發人員指南中的更新影像集中繼資料。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考UpdateImageSetMetadata中的。

Java

適用於 Java 2.x 的 SDK

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

           UpdateImageSetMetadataResponse response =
medicalImagingClient.updateImageSetMetadata(updateImageSetMetadataRequest);

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

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);
```

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);
```

使用案例 #3: 移除執行個體。

```
            final String removeInstance = """
                    {
```

```
                    "SchemaVersion": 1.1,
                    "Study": {
                      "Series": {
                        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1":
{
                          "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                          }
                        }
                      }
                    }
                  }
                  """;
          MetadataUpdates metadataRemoveUpdates = MetadataUpdates.builder()
                  .dicomUpdates(DICOMUpdates.builder()
                          .removableAttributes(SdkBytes.fromByteBuffer(
                                  ByteBuffer.wrap(removeInstance
                                          .getBytes(StandardCharsets.UTF_8))))
                          .build())
                  .build();

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

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考UpdateImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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

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

使用案例 #1: 插入或更新屬性。

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

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

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

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);
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考[UpdateImageSetMetadata](#)中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

Python

適用於 Python (Boto3) 的 SDK

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


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

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

下面的代碼實例化對 MedicalImagingWrapper 象。

```
    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
            )
```

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
            )
```

使用案例 #3: 移除執行個體。

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

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                                "Instances": {

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

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

- 如需 API 的詳細資訊，請參閱AWS 開發套件UpdateImageSetMetadata中的 Python (博托 3) API 參考。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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 SDK 的案例

下列程式碼範例說明如何在 AWS SDK 中 HealthImaging實作常見案例。這些案例會示範如何透過在其中呼叫多個函式來完成特定工作 HealthImaging。每個案例都包含一個連結 GitHub，您可以在其中找到如何設定和執行程式碼的指示。

範例

- 使用 AWS SDK 開始使用影 HealthImaging 像集和影像框
- 使用 AWS SDK 標記 HealthImaging 資料倉庫
- 使用 AWS SDK 標記 HealthImaging 影像集

使用 AWS SDK 開始使用影 HealthImaging 像集和影像框

下列程式碼範例說明如何在 HealthImaging中匯入 DICOM 檔案和下載影像框。

實作結構為工作流程命令列應用程式。

- 設定 DICOM 匯入的資源。
- 將 DICOM 檔案匯入資料倉庫。
- 擷取匯入工作的影像集 ID。
- 擷取影像集的影像框 ID。
- 下載，解碼和驗證圖像幀。
- 清理資源。

C++

適用於 C++ 的 SDK

使用必要的資源創建一個 AWS CloudFormation 堆棧。

```
   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,
                                               const Aws::String
 &outputBucketName,
                                               const Aws::String
 &outputDirectory,
                                               const Aws::String &roleArn,
                                               Aws::String &importJobId,
                                               const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    bool result = false;
    if (startDICOMImportJob(dataStoreID, inputBucketName, inputDirectory,
                            outputBucketName, outputDirectory, roleArn,
 importJobId,
                            clientConfiguration)) {
        std::cout << "DICOM import job started with job ID " << importJobId <<
 "."
                  << std::endl;
        result = waitImportJobCompleted(dataStoreID, importJobId,
 clientConfiguration);
        if (result) {
            std::cout << "DICOM import job completed." << std::endl;

        }
    }

    return result;
}

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

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

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

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

    return startDICOMImportJobOutcome.IsSuccess();
}


//! Routine which waits for a DICOM import job to complete.
/*!
 * @param dataStoreID: The HealthImaging data store ID.
 * @param importJobId: The import job ID.
 * @param clientConfiguration : Aws client configuration.
 * @return  bool: Function succeeded.
 */
```

```
bool AwsDoc::Medical_Imaging::waitImportJobCompleted(const Aws::String
 &datastoreID,
                                                     const Aws::String
 &importJobId,
                                                     const
 Aws::Client::ClientConfiguration &clientConfiguration) {

    Aws::MedicalImaging::Model::JobStatus jobStatus =
 Aws::MedicalImaging::Model::JobStatus::IN_PROGRESS;
    while (jobStatus == Aws::MedicalImaging::Model::JobStatus::IN_PROGRESS) {
        std::this_thread::sleep_for(std::chrono::seconds(1));

        Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
 getDicomImportJobOutcome = getDICOMImportJob(
                datastoreID, importJobId,
                clientConfiguration);

        if (getDicomImportJobOutcome.IsSuccess()) {
            jobStatus =
 getDicomImportJobOutcome.GetResult().GetJobProperties().GetJobStatus();

            std::cout << "DICOM import job status: " <<

 Aws::MedicalImaging::Model::JobStatusMapper::GetNameForJobStatus(
                              jobStatus) << std::endl;
        }
        else {
            std::cerr << "Failed to get import job status because "
                      << getDicomImportJobOutcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    }

    return jobStatus == Aws::MedicalImaging::Model::JobStatus::COMPLETED;
}

//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
  \param clientConfig: Aws client configuration.
  \return GetDICOMImportJobOutcome: The import job outcome.
*/
```

```
Aws::MedicalImaging::Model::GetDICOMImportJobOutcome
AwsDoc::Medical_Imaging::getDICOMImportJob(const Aws::String &dataStoreID,
                                           const Aws::String &importJobID,
                                           const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetDICOMImportJobRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetJobId(importJobID);
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome outcome =
 client.GetDICOMImportJob(
            request);
    if (!outcome.IsSuccess()) {
        std::cerr << "GetDICOMImportJob error: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome;
}
```

取得 DICOM 匯入工作所建立的影像集。

```
bool
AwsDoc::Medical_Imaging::getImageSetsForDicomImportJob(const Aws::String
 &datastoreID,
                                                       const Aws::String
 &importJobId,
                                                       Aws::Vector<Aws::String>
 &imageSets,
                                                       const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::MedicalImaging::Model::GetDICOMImportJobOutcome getDicomImportJobOutcome
 = getDICOMImportJob(
            datastoreID, importJobId, clientConfiguration);
    bool result = false;
    if (getDicomImportJobOutcome.IsSuccess()) {
        auto outputURI =
 getDicomImportJobOutcome.GetResult().GetJobProperties().GetOutputS3Uri();
        Aws::Http::URI uri(outputURI);
        const Aws::String &bucket = uri.GetAuthority();
        Aws::String key = uri.GetPath();
```

```
        Aws::S3::S3Client s3Client(clientConfiguration);
        Aws::S3::Model::GetObjectRequest objectRequest;
        objectRequest.SetBucket(bucket);
        objectRequest.SetKey(key + "/" + IMPORT_JOB_MANIFEST_FILE_NAME);

        auto getObjectOutcome = s3Client.GetObject(objectRequest);
        if (getObjectOutcome.IsSuccess()) {
            auto &data = getObjectOutcome.GetResult().GetBody();

            std::stringstream stringStream;
            stringStream << data.rdbuf();

            try {
                // Use JMESPath to extract the image set IDs.
                // https://jmespath.org/specification.html
                std::string jmesPathExpression =
"jobSummary.imageSetsSummary[].imageSetId";
                jsoncons::json doc = jsoncons::json::parse(stringStream.str());

                jsoncons::json imageSetsJson = jsoncons::jmespath::search(doc,

jmesPathExpression);\
                for (auto &imageSet: imageSetsJson.array_range()) {
                    imageSets.push_back(imageSet.as_string());
                }

                result = true;
            }
            catch (const std::exception &e) {
                std::cerr << e.what() << '\n';
            }

        }
        else {
            std::cerr << "Failed to get object because "
                      << getObjectOutcome.GetError().GetMessage() << std::endl;
        }

    }
    else {
        std::cerr << "Failed to get import job status because "
                  << getDicomImportJobOutcome.GetError().GetMessage() <<
std::endl;
    }
```

```
    return result;
}
```

取得影像集的影像框資訊。

```
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 outputFilePath: The path where the metadata will be stored as gzipped
 json.
  \param clientConfig: Aws client configuration.
  \\return bool: Function succeeded.
*/
bool AwsDoc::Medical_Imaging::getImageSetMetadata(const Aws::String &dataStoreID,
                                                  const Aws::String &imageSetID,
                                                  const Aws::String &versionID,
                                                  const Aws::String
 &outputFilePath,
                                                  const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::Model::GetImageSetMetadataRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    if (!versionID.empty()) {
        request.SetVersionId(versionID);
    }
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::GetImageSetMetadataOutcome outcome =
 client.GetImageSetMetadata(
            request);
    if (outcome.IsSuccess()) {
        std::ofstream file(outputFilePath, std::ios::binary);
        auto &metadata = outcome.GetResult().GetImageSetMetadataBlob();
        file << metadata.rdbuf();
    }
    else {
```

```
        std::cerr << "Failed to get image set metadata: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

下載，解碼和驗證圖像幀。

```
bool AwsDoc::Medical_Imaging::downloadDecodeAndCheckImageFrames(
        const Aws::String &dataStoreID,
        const Aws::Vector<ImageFrameInfo> &imageFrames,
        const Aws::String &outDirectory,
        const Aws::Client::ClientConfiguration &clientConfiguration) {

    Aws::Client::ClientConfiguration clientConfiguration1(clientConfiguration);
    clientConfiguration1.executor =
 Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>(
            "executor", 25);
    Aws::MedicalImaging::MedicalImagingClient medicalImagingClient(
            clientConfiguration1);

    Aws::Utils::Threading::Semaphore semaphore(0, 1);
    std::atomic<size_t> count(imageFrames.size());

    bool result = true;
    for (auto &imageFrame: imageFrames) {
        Aws::MedicalImaging::Model::GetImageFrameRequest getImageFrameRequest;
        getImageFrameRequest.SetDatastoreId(dataStoreID);
        getImageFrameRequest.SetImageSetId(imageFrame.mImageSetId);

        Aws::MedicalImaging::Model::ImageFrameInformation imageFrameInformation;
        imageFrameInformation.SetImageFrameId(imageFrame.mImageFrameId);
        getImageFrameRequest.SetImageFrameInformation(imageFrameInformation);

        auto getImageFrameAsyncLambda = [&semaphore, &result, &count, imageFrame,
 outDirectory](
                const Aws::MedicalImaging::MedicalImagingClient *client,
                const Aws::MedicalImaging::Model::GetImageFrameRequest &request,
                Aws::MedicalImaging::Model::GetImageFrameOutcome outcome,
                const std::shared_ptr<const Aws::Client::AsyncCallerContext>
 &context) {
```

```
                if (!handleGetImageFrameResult(outcome, outDirectory,
 imageFrame)) {
                    std::cerr << "Failed to download and convert image frame: "
                              << imageFrame.mImageFrameId << " from image set: "
                              << imageFrame.mImageSetId << std::endl;
                    result = false;
                }

                count--;
                if (count <= 0) {

                    semaphore.ReleaseAll();
                }
        }; // End of 'getImageFrameAsyncLambda' lambda.

        medicalImagingClient.GetImageFrameAsync(getImageFrameRequest,
                                                getImageFrameAsyncLambda);
    }

    if (count > 0) {
        semaphore.WaitOne();
    }

    if (result) {
        std::cout << imageFrames.size() << " image files were downloaded."
                  << std::endl;
    }

    return result;
}

bool AwsDoc::Medical_Imaging::decodeJPHFileAndValidateWithChecksum(
        const Aws::String &jphFile,
        uint32_t crc32Checksum) {
    opj_image_t *outputImage = jphImageToOpjBitmap(jphFile);
    if (!outputImage) {
        return false;
    }

    bool result = true;
    if (!verifyChecksumForImage(outputImage, crc32Checksum)) {
        std::cerr << "The checksum for the image does not match the expected
 value."
```

```
                  << std::endl;
        std::cerr << "File :" << jphFile << std::endl;
        result = false;
    }

    opj_image_destroy(outputImage);

    return result;
}

opj_image *
AwsDoc::Medical_Imaging::jphImageToOpjBitmap(const Aws::String &jphFile) {
    opj_stream_t *inFileStream = nullptr;
    opj_codec_t *decompressorCodec = nullptr;
    opj_image_t *outputImage = nullptr;
    try {
        std::shared_ptr<opj_dparameters> decodeParameters =
 std::make_shared<opj_dparameters>();
        memset(decodeParameters.get(), 0, sizeof(opj_dparameters));

        opj_set_default_decoder_parameters(decodeParameters.get());

        decodeParameters->decod_format = 1; // JP2 image format.
        decodeParameters->cod_format = 2; // BMP image format.

        std::strncpy(decodeParameters->infile, jphFile.c_str(),
                     OPJ_PATH_LEN);

        inFileStream = opj_stream_create_default_file_stream(
                decodeParameters->infile, true);
        if (!inFileStream) {
            throw std::runtime_error(
                    "Unable to create input file stream for file '" + jphFile +
 "'.");
        }

        decompressorCodec = opj_create_decompress(OPJ_CODEC_JP2);
        if (!decompressorCodec) {
            throw std::runtime_error("Failed to create decompression codec.");
        }

        int decodeMessageLevel = 1;
        if (!setupCodecLogging(decompressorCodec, &decodeMessageLevel)) {
            std::cerr << "Failed to setup codec logging." << std::endl;
```

```
        }

        if (!opj_setup_decoder(decompressorCodec, decodeParameters.get())) {
            throw std::runtime_error("Failed to setup decompression codec.");
        }
        if (!opj_codec_set_threads(decompressorCodec, 4)) {
            throw std::runtime_error("Failed to set decompression codec
threads.");
        }

        if (!opj_read_header(inFileStream, decompressorCodec, &outputImage)) {
            throw std::runtime_error("Failed to read header.");
        }

        if (!opj_decode(decompressorCodec, inFileStream,
                        outputImage)) {
            throw std::runtime_error("Failed to decode.");
        }

        if (DEBUGGING) {
            std::cout << "image width : " << outputImage->x1 - outputImage->x0
                      << std::endl;
            std::cout << "image height : " << outputImage->y1 - outputImage->y0
                      << std::endl;
            std::cout << "number of channels: " << outputImage->numcomps
                      << std::endl;
            std::cout << "colorspace : " << outputImage->color_space <<
std::endl;
        }

    } catch (const std::exception &e) {
        std::cerr << e.what() << std::endl;
        if (outputImage) {
            opj_image_destroy(outputImage);
            outputImage = nullptr;
        }
    }
    if (inFileStream) {
        opj_stream_destroy(inFileStream);
    }
    if (decompressorCodec) {
        opj_destroy_codec(decompressorCodec);
    }
```

```
    return outputImage;
}

//! Template function which converts a planar image bitmap to an interleaved
 image bitmap and
//! then verifies the checksum of the bitmap.
/*!
 * @param image: The OpenJPEG image struct.
 * @param crc32Checksum: The CRC32 checksum.
 * @return  bool: Function succeeded.
 */
template<class myType>
bool verifyChecksumForImageForType(opj_image_t *image, uint32_t crc32Checksum) {
    uint32_t width = image->x1 - image->x0;
    uint32_t height = image->y1 - image->y0;
    uint32_t numOfChannels = image->numcomps;

    // Buffer for interleaved bitmap.
    std::vector<myType> buffer(width * height * numOfChannels);

    // Convert planar bitmap to interleaved bitmap.
    for (uint32_t channel = 0; channel < numOfChannels; channel++) {
        for (uint32_t row = 0; row < height; row++) {
            uint32_t fromRowStart = row / image->comps[channel].dy * width /
                                    image->comps[channel].dx;
            uint32_t toIndex = (row * width) * numOfChannels + channel;

            for (uint32_t col = 0; col < width; col++) {
                uint32_t fromIndex = fromRowStart + col / image-
>comps[channel].dx;

                buffer[toIndex] = static_cast<myType>(image-
>comps[channel].data[fromIndex]);

                toIndex += numOfChannels;
            }
        }
    }

    // Verify checksum.
    boost::crc_32_type crc32;
    crc32.process_bytes(reinterpret_cast<char *>(buffer.data()),
                        buffer.size() * sizeof(myType));
```

```
    bool result = crc32.checksum() == crc32Checksum;
    if (!result) {
        std::cerr << "verifyChecksumForImage, checksum mismatch, expected - "
                  << crc32Checksum << ", actual - " << crc32.checksum()
                  << std::endl;
    }

    return result;
}

//! Routine which verifies the checksum of an OpenJPEG image struct.
/*!
 * @param image: The OpenJPEG image struct.
 * @param crc32Checksum: The CRC32 checksum.
 * @return  bool: Function succeeded.
 */
bool AwsDoc::Medical_Imaging::verifyChecksumForImage(opj_image_t *image,
                                                     uint32_t crc32Checksum) {
    uint32_t channels = image->numcomps;
    bool result = false;
    if (0 < channels) {
        // Assume the precision is the same for all channels.
        uint32_t precision = image->comps[0].prec;
        bool signedData = image->comps[0].sgnd;
        uint32_t bytes = (precision + 7) / 8;

        if (signedData) {
            switch (bytes) {
                case 1 :
                    result = verifyChecksumForImageForType<int8_t>(image,

 crc32Checksum);
                    break;
                case 2 :
                    result = verifyChecksumForImageForType<int16_t>(image,

 crc32Checksum);
                    break;
                case 4 :
                    result = verifyChecksumForImageForType<int32_t>(image,

 crc32Checksum);
                    break;
                default:
```

```
                    std::cerr
                            << "verifyChecksumForImage, unsupported data type,
signed bytes - "
                            << bytes << std::endl;
                    break;
            }
        }
        else {
            switch (bytes) {
                case 1 :
                    result = verifyChecksumForImageForType<uint8_t>(image,

crc32Checksum);
                    break;
                case 2 :
                    result = verifyChecksumForImageForType<uint16_t>(image,

crc32Checksum);
                    break;
                case 4 :
                    result = verifyChecksumForImageForType<uint32_t>(image,

crc32Checksum);
                    break;
                default:
                    std::cerr
                            << "verifyChecksumForImage, unsupported data type,
unsigned bytes - "
                            << bytes << std::endl;
                    break;
            }
        }

        if (!result) {
            std::cerr << "verifyChecksumForImage, error bytes " << bytes
                      << " signed "
                      << signedData << std::endl;
        }
    }
    else {
        std::cerr << "'verifyChecksumForImage', no channels in the image."
                  << std::endl;
    }
    return result;
```

```
}
```

清理資源。

```
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《AWS SDK for C++ API 參考》中的下列主題。

- DeleteImageSet
- GetDicom ImportJob
- GetImageFrame
- GetImageSetMetadata
- SearchImageSets
- 开始迪克 ImportJob

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

index.js-協調步驟。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
```

```
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
```

```
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js-部署資源。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import fs from "node:fs/promises";
```

```
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

```
export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
```

```
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName == state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
```

```
);
```

dataset-steps.js-複製 DICOM 文件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
```

```
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);
```

```
    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js-開始匯入資料存放區。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
```

```
/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
```

```
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js-取得影像集 ID。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
```

```
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }
[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
```

```
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js-獲取圖像幀 ID。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
```

```
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
```

```
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }
```

```
    return output;
  },
  { slow: false },
);
```

verify-steps.js-驗證圖像幀。使用「AWS HealthImaging 像素資料驗證」程式庫進行驗證。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
```

```
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (/** @type {State} */ state) => {
```

```
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid}
and sop ${sopInstanceUid}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(
                    `Verification tool exited with code ${code} for image set
${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
```

```
      }
    }
  },
);
```

clean-up-steps.js-摧毀資源。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
```

```
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
```

```
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
```

```
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- 如需 API 詳細資訊，請參閱《AWS SDK for JavaScript API 參考》中的下列主題。
  - DeleteImageSet
  - GetDicom ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - 开始迪克 ImportJob

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

使用必要的資源創建一個 AWS CloudFormation 堆棧。

```
    def deploy(self):
        """
        Deploys prerequisite resources used by the scenario. The resources are
        defined in the associated `setup.yaml` AWS CloudFormation script and are
deployed
        as a CloudFormation stack, so they can be easily managed and destroyed.
        """

        print("\t\tLet's deploy the stack for resource creation.")
        stack_name = q.ask("\t\tEnter a name for the stack: ", q.non_empty)

        data_store_name = q.ask(
            "\t\tEnter a name for the Health Imaging Data Store: ", q.non_empty
        )
```

```
        account_id = boto3.client("sts").get_caller_identity()["Account"]

        with open(
            "../../../../workflows/healthimaging_image_sets/resources/
cfn_template.yaml"
        ) as setup_file:
            setup_template = setup_file.read()
        print(f"\t\tCreating {stack_name}.")
        stack = self.cf_resource.create_stack(
            StackName=stack_name,
            TemplateBody=setup_template,
            Capabilities=["CAPABILITY_NAMED_IAM"],
            Parameters=[
                {
                    "ParameterKey": "datastoreName",
                    "ParameterValue": data_store_name,
                },
                {
                    "ParameterKey": "userAccountID",
                    "ParameterValue": account_id,
                },
            ],
        )
        print("\t\tWaiting for stack to deploy. This typically takes a minute or
 two.")
        waiter = self.cf_resource.meta.client.get_waiter("stack_create_complete")
        waiter.wait(StackName=stack.name)
        stack.load()
        print(f"\t\tStack status: {stack.stack_status}")

        outputs_dictionary = {
            output["OutputKey"]: output["OutputValue"] for output in
 stack.outputs
        }
        self.input_bucket_name = outputs_dictionary["BucketName"]
        self.output_bucket_name = outputs_dictionary["BucketName"]
        self.role_arn = outputs_dictionary["RoleArn"]
        self.data_store_id = outputs_dictionary["DatastoreID"]
        return stack
```

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 Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def start_dicom_import_job(
        self,
        data_store_id,
        input_bucket_name,
        input_directory,
        output_bucket_name,
        output_directory,
        role_arn,
    ):
        """
        Routine which starts a HealthImaging import job.

        :param data_store_id: The HealthImaging data store ID.
        :param input_bucket_name: The name of the Amazon S3 bucket containing the
 DICOM files.
        :param input_directory: The directory in the S3 bucket containing the
 DICOM files.
```

```
        :param output_bucket_name: The name of the S3 bucket for the output.
        :param output_directory: The directory in the S3 bucket to store the
 output.
        :param role_arn: The ARN of the IAM role with permissions for the import.
        :return: The job ID of the import.
        """

        input_uri = f"s3://{input_bucket_name}/{input_directory}/"
        output_uri = f"s3://{output_bucket_name}/{output_directory}/"
        try:
            job = self.medical_imaging_client.start_dicom_import_job(
                jobName="examplejob",
                datastoreId=data_store_id,
                dataAccessRoleArn=role_arn,
                inputS3Uri=input_uri,
                outputS3Uri=output_uri,
            )
        except ClientError as err:
            logger.error(
                "Couldn't start DICOM import job. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return job["jobId"]
```

取得 DICOM 匯入工作所建立的影像集。

```


class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client
```

```
    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_sets_for_dicom_import_job(self, datastore_id, import_job_id):
        """
        Retrieves the image sets created for an import job.

        :param datastore_id: The HealthImaging data store ID
        :param import_job_id: The import job ID
        :return: List of image set IDs
        """

        import_job = self.medical_imaging_client.get_dicom_import_job(
            datastoreId=datastore_id, jobId=import_job_id
        )

        output_uri = import_job["jobProperties"]["outputS3Uri"]

        bucket = output_uri.split("/")[2]
        key = "/".join(output_uri.split("/")[3:])

        # Try to get the manifest.
        retries = 3
        while retries > 0:
            try:
                obj = self.s3_client.get_object(
                    Bucket=bucket, Key=key + "job-output-manifest.json"
                )
                body = obj["Body"]
                break
            except ClientError as error:
                retries = retries - 1
                time.sleep(3)
        try:
            data = json.load(body)
            expression =
jmespath.compile("jobSummary.imageSetsSummary[].imageSetId")
            image_sets = expression.search(data)
        except json.decoder.JSONDecodeError as error:
```

```
            image_sets = import_job["jobProperties"]

        return image_sets


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

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

取得影像集的影像框資訊。

```


class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""
```

```
    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def get_image_frames_for_image_set(self, datastore_id, image_set_id,
out_directory):
        """
        Get the image frames for an image set.

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param out_directory: The directory to save the file.
        :return: The image frames.
        """
        image_frames = []
        file_name = os.path.join(out_directory,
f"{image_set_id}_metadata.json.gzip")
        file_name = file_name.replace("/", "\\\\")
        self.get_image_set_metadata(file_name, datastore_id, image_set_id)
        try:
            with gzip.open(file_name, "rb") as f_in:
                doc = json.load(f_in)
            instances = jmespath.search("Study.Series.*.Instances[].*[]", doc)
            for instance in instances:
                rescale_slope = jmespath.search("DICOM.RescaleSlope", instance)
                rescale_intercept = jmespath.search("DICOM.RescaleIntercept",
instance)
                image_frames_json = jmespath.search("ImageFrames[][]", instance)
                for image_frame in image_frames_json:
                    checksum_json = jmespath.search(
                        "max_by(PixelDataChecksumFromBaseToFullResolution,
&Width)",
```

```
                        image_frame,
                    )
                    image_frame_info = {
                        "imageSetId": image_set_id,
                        "imageFrameId": image_frame["ID"],
                        "rescaleIntercept": rescale_intercept,
                        "rescaleSlope": rescale_slope,
                        "minPixelValue": image_frame["MinPixelValue"],
                        "maxPixelValue": image_frame["MaxPixelValue"],
                        "fullResolutionChecksum": checksum_json["Checksum"],
                    }
                    image_frames.append(image_frame_info)
            return image_frames
        except TypeError:
            return {}
        except ClientError as err:
            logger.error(
                "Couldn't get image frames for image set. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        return image_frames


    def get_image_set_metadata(
        self, metadata_file, datastore_id, image_set_id, version_id=None
    ):
        """
        Get the metadata of an image set.

        :param metadata_file: The file to store the JSON gzipped metadata.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The version of the image set.
        """

        try:
            if version_id:
                image_set_metadata =
self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
```

```
                )
            else:
                image_set_metadata =
 self.medical_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id, datastoreId=datastore_id
                )
            with open(metadata_file, "wb") as f:
                for chunk in
 image_set_metadata["imageSetMetadataBlob"].iter_chunks():
                    if chunk:
                        f.write(chunk)

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

下載，解碼和驗證圖像幀。

```
class MedicalImagingWrapper:
    """Encapsulates Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)
```

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

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


    def download_decode_and_check_image_frames(
        self, data_store_id, image_frames, out_directory
    ):
        """
        Downloads image frames, decodes them, and uses the checksum to validate
        the decoded images.

        :param data_store_id: The HealthImaging data store ID.
        :param image_frames: A list of dicts containing image frame information.
        :param out_directory: A directory for the downloaded images.
        :return: True if the function succeeded; otherwise, False.
        """
        total_result = True
```

```
        for image_frame in image_frames:
            image_file_path = f"{out_directory}/
image_{image_frame['imageFrameId']}.jph"
            self.get_pixel_data(
                image_file_path,
                data_store_id,
                image_frame["imageSetId"],
                image_frame["imageFrameId"],
            )

            image_array = self.jph_image_to_opj_bitmap(image_file_path)
            crc32_checksum = image_frame["fullResolutionChecksum"]
            # Verify checksum.
            crc32_calculated = zlib.crc32(image_array)
            image_result = crc32_checksum == crc32_calculated
            print(
                f"\t\tImage checksum verified for {image_frame['imageFrameId']}:
 {image_result }"
            )
            total_result = total_result and image_result
        return total_result

    @staticmethod
    def jph_image_to_opj_bitmap(jph_file):
        """
        Decode the image to a bitmap using an OPENJPEG library.
        :param jph_file: The file to decode.
        :return: The decoded bitmap as an array.
        """
        # Use format 2 for the JPH file.
        params = openjpeg.utils.get_parameters(jph_file, 2)
        print(f"\n\t\tImage parameters for {jph_file}: \n\t\t{params}")

        image_array = openjpeg.utils.decode(jph_file, 2)

        return image_array
```

清理資源。

```
    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 Amazon HealthImaging functionality."""

    def __init__(self, medical_imaging_client, s3_client):
        """
        :param medical_imaging_client: A Boto3 Amazon MedicalImaging client.
        :param s3_client: A Boto3 S3 client.
        """
```

```
        self.medical_imaging_client = medical_imaging_client
        self.s3_client = s3_client

    @classmethod
    def from_client(cls):
        medical_imaging_client = boto3.client("medical-imaging")
        s3_client = boto3.client("s3")
        return cls(medical_imaging_client, s3_client)


    def search_image_sets(self, datastore_id, search_filter):
        """
        Search for image sets.

        :param datastore_id: The ID of the data store.
        :param search_filter: The search filter.
            For example: {"filters" : [{ "operator": "EQUAL", "values":
[{"DICOMPatientId": "3524578"}]}]}.
        :return: The list of image sets.
        """
        try:
            paginator =
self.medical_imaging_client.get_paginator("search_image_sets")
            page_iterator = paginator.paginate(
                datastoreId=datastore_id, searchCriteria=search_filter
            )
            metadata_summaries = []
            for page in page_iterator:
                metadata_summaries.extend(page["imageSetsMetadataSummaries"])
        except ClientError as err:
            logger.error(
                "Couldn't search image sets. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return metadata_summaries


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

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

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考資料》中的下列主題。
  - DeleteImageSet
  - GetDicom ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - 开始迪克 ImportJob

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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 資料倉庫

下列程式碼範例顯示如何標記 HealthImaging 資料倉庫。

Java

適用於 Java 2.x 的 SDK

為資料倉庫加上標籤。

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

用於標記資源的實用程序功能。

```
   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);
       }
   }
```

列示資料倉庫的標籤。

```
                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功能。

```
   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;
   }
```

取消標籤資料倉庫。

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

取消標記資源的公用程式功能。

```
   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**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

為資料倉庫加上標籤。

```
try {
  const datastoreArn =
```

```
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(datastoreArn, tags);
  } catch (e) {
    console.log(e);
  }
```

用於標記資源的實用程序功能。

```
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;
```

```
};
```

列示資料倉庫的標籤。

```
  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功能。

```
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;
};
```

取消標籤資料倉庫。

```
  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);
  }
```

取消標記資源的公用程式功能。

```
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《AWS SDK for JavaScript API 參考》中的下列主題。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

為資料倉庫加上標籤。

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

用於標記資源的實用程序功能。

```
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
```

列示資料倉庫的標籤。

```
    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功能。

```
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"]
```

取消標籤資料倉庫。

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

取消標記資源的公用程式功能。

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考資料》中的下列主題。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 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 影像集

下列程式碼範例顯示如何標記 HealthImaging 影像集。

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"));
```

用於標記資源的實用程序功能。

```
    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功能。

```
   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"));
```

取消標記資源的公用程式功能。

```
   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**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

JavaScript

適用於 JavaScript (v3) 的開發套件

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);
  }
```

用於標記資源的實用程序功能。

```
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功能。

```
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);
  }
```

取消標記資源的公用程式功能。

```
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《AWS SDK for JavaScript API 參考》中的下列主題。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

Python

適用於 Python (Boto3) 的 SDK

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"})
```

用於標記資源的實用程序功能。

```
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功能。

```
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"])
```

取消標記資源的公用程式功能。

```
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
```

下面的代碼實例化對 MedicalImagingWrapper 象。

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

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考資料》中的下列主題。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭 HealthImaging 配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

中的安全性 AWS HealthImaging

雲安全 AWS 是最高的優先級。身為 AWS 客戶，您可以從資料中心和網路架構中獲益，這些架構是為了滿足對安全性最敏感的組織的需求而建置的。

安全是 AWS 與您之間共同的責任。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：

- 雲端安全性 — AWS 負責保護中執行 AWS 服務的基礎架構 AWS 雲端。 AWS 還為您提供可以安全使用的服務。若要深入瞭解適用於的規範遵循計劃 AWS HealthImaging，請參閱合規計劃的AWS 服務範圍範圍)。
- 雲端中的安全性 — 您的責任取決於您使用的 AWS 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。

本文件可協助您瞭解如何在使用時套用共同責任模型 HealthImaging。下列主題說明如何設定 HealthImaging 以符合安全性與合規性目標。您也會學到如何使用其他可協助您監控和保護 HealthImaging 資源的 AWS 服務。

主題

- AWS 中的資料保護 HealthImaging
- AWS 的 Identity and Access Management HealthImaging
- AWS 中的記錄和監控 HealthImaging
- 適用於 AWS 的合規驗證 HealthImaging
- AWS 中的彈性 HealthImaging
- AWS 中的基礎設施安全 HealthImaging
- 使用建立 AWS HealthImaging 資源 AWS CloudFormation
- AWS HealthImaging 和介面 VPC 端端點 ()AWS PrivateLink
- 跨帳戶匯入 AWS HealthImaging

AWS 中的資料保護 HealthImaging

AWS 共同責任模型適用於 AWS 中的資料保護 HealthImaging。如此模型中所述， AWS 負責保護執行所有 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，我們建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 登入資料並設定個別使用者。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 資源進行通訊。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。
- 使用 AWS 加密解決方案，以及其中的所有默認安全控制 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。
- 如果您在透過命令列介面或 API 存取時需要經 AWS 過 FIPS 140-2 驗證的加密模組，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) 140-2 概觀。

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。這包括當您使用主控台、API HealthImaging 或 AWS SDK 時 AWS 服務 使用或其他使用時。 AWS CLI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

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 服務中提供的靜態資料加密功能
- 彈性的金鑰管理選項 AWS Key Management Service，包括您可以選擇是否要 AWS 管理加密金鑰或完全掌控自己的金鑰。
- AWS 擁有的 AWS KMS 加密金鑰
- 針對 Amazon SQS 使用伺服器端加密 (SSE) 傳輸敏感資料的加密訊息佇列

此外，還 AWS 提供 API 供您將加密和資料保護與您在 AWS 環境中開發或部署的任何服務整合。

靜態加密

HealthImaging 默認情況下提供加密，通過使用服務擁有的密 AWS KMS 鑰來保護靜態客戶的敏感數據。

傳輸中加密

HealthImaging 使用 TLS 1.2 加密透過公用端點和後端服務傳輸中的資料。

金鑰管理

AWS KMS 金鑰 (KMS 金鑰) 是中的主要資源 AWS Key Management Service。您也可以產生資料金鑰以供在外部使用 AWS KMS。

AWS 擁有的 KMS 金鑰

HealthImaging 默認情況下使用這些密鑰來自動加密潛在的敏感信息，例如個人身份或靜態私人 Health 信息（PHI）數據。 AWS 擁有的 KMS 金鑰不會儲存在您的帳戶中。它們是 AWS 擁有和管理以在多個 AWS 帳戶中使用的 KMS 金鑰集合的一部分。 AWS 服務可以使用 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，或稽核其使用情況。但是，您無需執行任何工作或更改任何程序即可保護加密數據的密鑰。

如果您使用 AWS 擁有的 KMS 金鑰，則不會向您收取月費或使用費，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

HealthImaging 支援使用您建立、擁有和管理的對稱客戶受管 KMS 金鑰，以針對現有 AWS 擁有的加密新增第二層加密。您可以完全控制此層加密，因此能執行以下任務：

- 建立和維護關鍵政策、IAM 政策和撥款
- 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 來追蹤代表您 HealthImaging 傳送給 AWS KMS 的要求。需 AWS KMS 支付額外費用。如需更多資訊，請參閱 AWS Key Management Service 開發人員指南中的客戶受管金鑰。

建立客戶管理的金鑰

您可以使用 AWS Management Console 或 AWS KMS API 建立對稱的客戶管理金鑰。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資 HealthImaging 源搭配使用，必須在金鑰原則中允許 kms: CreateGrant 作業。這會將授權新增至客戶受管金鑰，該金鑰可控制對指定 KMS 金鑰的存取權，讓使用者能夠存取授與作業所 HealthImaging 需的權限。如需詳細資訊，請參閱AWS Key Management Service 開發人員指南 AWS KMS中的授權。

若要將客戶受管 KMS 金鑰與您的 HealthImaging 資源搭配使用，必須在金鑰原則中允許下列 API 作業：

- `kms:DescribeKey`提供驗證金鑰所需的客戶管理金鑰詳細資料。這是所有操作都必需的。
- `kms:GenerateDataKey`提供對所有寫入作業靜態加密資源的存取權。
- `kms:Decrypt`提供對加密資源的讀取或搜尋作業的存取權。
- `kms:ReEncrypt*`提供重新加密資源的存取權。

以下是政策陳述式範例，可讓使用者建立以該金鑰加密的 HealthImaging 資料存放區並與之互動：

```
{
    "Sid": "Allow access to create data stores and perform CRUD and search in
 HealthImaging",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "medical-imaging.amazonaws.com"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
```

```
            "kms:EncryptionContext:kms-arn": "arn:aws:kms:us-east-1:123456789012:key/
bec71d48-3462-4cdd-9514-77a7226e001f",
            "kms:EncryptionContext:aws:medical-imaging:datastoreId": "datastoreId"
        }
    }
}
```

使用客戶受管 KMS 金鑰所需的 IAM 許可

使用客戶管理的 KMS 金鑰建立啟用 AWS KMS 加密的資料存放區時，金鑰政策和 IAM 政策的使用者或建立 HealthImaging 資料存放區的角色都需要許可。

如需關鍵政策的詳細資訊，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`kms:CreateGrant`、`kms:GenerateDataKey``kms:RetireGrant``kms:Decrypt`、和的許可`kms:ReEncrypt*`，以及 AWS 必要的許可 HealthImaging。

如何 HealthImaging 使用贈款 AWS KMS

HealthImaging 需要授權才能使用客戶管理的 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，CreateGrant請將請求傳送至以代表您建 HealthImaging 立授權 AWS KMS。中的授 HealthImaging 權 AWS KMS 用於授與客戶帳戶中 KMS 金鑰的存取權。

代表您建 HealthImaging 立的授權不應該撤銷或淘汰。如果您撤銷或淘汰授與使用您帳戶中 AWS KMS 金鑰之 HealthImaging 權限的授權，則 HealthImaging 無法存取此資料、加密推送至資料存放區的新影像資源，或在提取資料存放區時將其解密。當您撤銷或淘汰的授權時 HealthImaging，變更會立即發生。若要撤銷存取權限，您應該刪除資料存放區，而不是撤銷授與。刪除資料存放區後，代表您 HealthImaging 淘汰授權。

監控您的加密金鑰 HealthImaging

使用客戶管理 CloudTrail 的 KMS 金鑰時，您可以使用來 AWS KMS 追蹤代表您 HealthImaging 傳送的要求。記錄檔中的記 CloudTrail 錄項目會顯示`medical-imaging.amazonaws.com`在`userAgent`欄位中，以清楚區分由提出的要求 HealthImaging。

下列範例為`CreateGrant`、`GenerateDataKeyDecrypt`、和監視呼叫的 AWS KMS 作業`DescribeKey`以存取由 HealthImaging 客戶管理金鑰加密之資料的 CloudTrail 事件。

以下說明如`CreateGrant`何使用允許存 HealthImaging 取客戶提供的 KMS 金鑰，以 HealthImaging 便使用該 KMS 金鑰加密所有靜態客戶資料。

使用者不需要建立自己的授權。 HealthImaging 將CreateGrant請求傳送至以下位置，以代表您建立授權 AWS KMS。中的贈款 AWS KMS 用於授予對客戶帳戶中 AWS KMS 密鑰的 HealthImaging 訪問權限。


```
{
    "Grants": [
        {
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "GenerateDataKeyWithoutPlaintext",
                "DescribeKey"
            ],
            "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
            "Name": "0a74e6ad2aa84b74a22fcd3efac1eaa8",
            "RetiringPrincipal": "AWS Internal",
            "GranteePrincipal": "AWS Internal",
            "GrantId":
 "0da169eb18ffd3da8c0eebc9e74b3839573eb87e1e0dce893bb544a34e8fbaaf",
            "IssuingAccount": "AWS Internal",
            "CreationDate": 1685050229.0,
            "Constraints": {
                "EncryptionContextSubset": {
                    "kms-arn": "arn:aws:kms:us-
west-2:824333766656:key/2fe3c119-792d-4b99-822f-b5841e1181d1"
                }
            }
        },
        {
            "Operations": [
                "GenerateDataKey",
                "CreateGrant",
                "RetireGrant",
                "DescribeKey"
            ],
            "KeyId": "arn:aws:kms:us-west-2:824333766656:key/2fe3c119-792d-4b99-822f-
b5841e1181d1",
            "Name": "2023-05-25T21:30:17",
            "RetiringPrincipal": "AWS Internal",
            "GranteePrincipal": "AWS Internal",
```

```
            "GrantId":
 "8229757abbb2019555ba64d200278cedac08e5a7147426536fcd1f4270040a31",
            "IssuingAccount": "AWS Internal",
            "CreationDate": 1685050217.0,
        }
    ]
}
```

下列範例說明如何使用以確GenerateDataKey保使用者在儲存資料之前擁有加密資料的必要權限。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLEUSER",
        "arn": "arn:aws:sts::111122223333:assumed-role/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLEKEYID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLEROLE",
                "arn": "arn:aws:iam::111122223333:role/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2021-06-30T21:17:06Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "medical-imaging.amazonaws.com"
    },
    "eventTime": "2021-06-30T21:17:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "medical-imaging.amazonaws.com",
    "userAgent": "medical-imaging.amazonaws.com",
    "requestParameters": {
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
```

```
    },
    "responseElements": null,
    "requestID": "EXAMPLE_ID_01",
    "eventID": "EXAMPLE_ID_02",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

下列範例顯示如何 HealthImaging 呼叫Decrypt作業，以使用儲存的加密資料金鑰存取加密的資料。

```
{
    "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"
            }
        },
        "invokedBy": "medical-imaging.amazonaws.com"
```

```
    },
    "eventTime": "2021-06-30T21:21:59Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "medical-imaging.amazonaws.com",
    "userAgent": "medical-imaging.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
    },
    "responseElements": null,
    "requestID": "EXAMPLE_ID_01",
    "eventID": "EXAMPLE_ID_02",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

下列範例顯示如何 HealthImaging 使用DescribeKey作業來驗證 AWS KMS 客戶擁有的 AWS KMS 金鑰是否處於可用狀態，並協助使用者疑難排解無法運作。

```
{
    "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，請參閱AWS Key Management Service 開發人員指南中的。

- AWS KMS 概念
- 安全性最佳做法 AWS KMS

網路流量隱私

在現場部署應用程式之間以 HealthImaging 及與 Amazon S3 之間 HealthImaging的流量都受到保護。預設情況下 HealthImaging ，和之間的流量 AWS Key Management Service 會使用 HTTPS。

- AWS HealthImaging 是在美國東部 (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，傳輸層安全性 (TLS) 會加密 HealthImaging 和 Amazon S3 之間的傳輸中物件，以及存取它的客戶應用程式之間 HealthImaging 的物件，您應該只允許使用 Amazon S3 儲存貯體 IAM 政策透過 HTTPS (TLS) 加密連線。`aws:SecureTransport condition`雖然 HealthImaging 目前使用公有端點存取 Amazon S3 儲存貯體中的資料，但這並不表示資料會周遊公用網際網路。 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 的 Identity and Access Management HealthImaging

AWS Identity and Access Management (IAM) 可協助管理員安全地控制 AWS 資源存取權。 AWS 服務 IAM 管理員控制哪些人可以通過身份驗證 (登入) 和授權 (具有權限) 來使用 HealthImaging 資源。IAM 是您可以使用的 AWS 服務 ，無需額外付費。

主題

- 物件
- 使用身分驗證
- 使用政策管理存取權
- AWS 如何與 IAM HealthImaging 搭配使用
- AWS 的身分型政策範例 HealthImaging
- AWS適用於 AWS 的受管政策 HealthImaging
- 疑難排解 AWS HealthImaging 身分和存取

物件

您使用 AWS Identity and Access Management (IAM) 的方式會有所不同，具體取決於您在進行的工作 HealthImaging。

服務使用者 — 如果您使用 HealthImaging 服務執行工作，則管理員會為您提供所需的認證和權限。當您使用更多 HealthImaging 功能來完成工作時，您可能需要其他權限。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。如果無法存取中的圖徵 HealthImaging，請參閱疑難排解 AWS HealthImaging 身分和存取。

服務管理員 — 如果您負責公司的 HealthImaging 資源，您可能擁有完整的存取權 HealthImaging。決定您的服務使用者應該存取哪些 HealthImaging 功能和資源是您的工作。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。若要進一步瞭解貴公司如何搭配使用 IAM HealthImaging，請參閱AWS 如何與 IAM HealthImaging 搭配使用。

IAM 管理員 — 如果您是 IAM 管理員，您可能想要瞭解如何撰寫政策來管理存取權限的詳細資訊 HealthImaging。若要檢視可在 IAM 中使用的 HealthImaging 基於身分的政策範例，請參閱。AWS 的身分型政策範例 HealthImaging

使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。 AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。 AWS IAM Identity Center (IAM 身分中心) 使用者、貴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識別的範例。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 AWS 用同盟存取時，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 AWS 帳戶的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的簽署 AWS API 請求。

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 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。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 使用者指南中的在 AWS中使用多重要素驗證 (MFA)。

AWS 帳戶 根使用者

建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取該帳戶中的所有資源 AWS 服務 和資源。此身分稱為 AWS 帳戶 root 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

最佳作法是要求人類使用者 (包括需要系統管理員存取權的使用者) 使用與身分識別提供者的同盟，才能使用臨時認證 AWS 服務 來存取。

聯合身分識別是來自企業使用者目錄的使用者、Web 身分識別提供者、Identi ty Center 目錄，或使用透過身分識別來源提供的認證進行存取 AWS 服務 的任何使用者。 AWS Directory Service同盟身分存取時 AWS 帳戶，他們會假設角色，而角色則提供臨時認證。

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，也可以連線並同步到自己身分識別來源中的一組使用者和群組，以便在所有應用程式 AWS 帳戶 和應用程式中使用。如需 IAM Identity Center 的相關資訊，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 使用者是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，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，但角色僅提供暫時憑證。如需進一步了解，請參閱 IAM 使用者指南中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以切換角色，在中暫時擔任 IAM 角色。 AWS Management Console 您可以透過呼叫 AWS CLI 或

AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法更多相關資訊，請參閱 IAM 使用者指南中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 – 若要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 (信任的委託人) 存取您帳戶中的資源。角色是授予跨帳戶存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源（而不是使用角色作為代理）。若要了解跨帳戶存取權角色和資源型政策間的差異，請參閱 IAM 使用者指南中的 IAM 角色與資源類型政策的差異。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。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。
  - 轉寄存取工作階段 (FAS) — 當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。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 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 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。如需更多資訊，請參閱 IAM 使用者指南中的建立角色以委派許可給 AWS 服務。
  - 服務連結角色 — 服務連結角色是連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。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 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。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 執行個體並提供給其所有應用程式，請建立連接至執行個體的執行個體設定檔。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暫時憑證。如需更多資訊，請參閱 IAM 使用者指南中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

若要了解是否要使用 IAM 角色或 IAM 使用者，請參閱 IAM 使用者指南中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS ，當與身分識別或資源相關聯時，會定義其權限。 AWS 當主參與者 (使用者、root 使用者或角色工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以 JSON 文件的形式儲存在中。如需 JSON 政策文件結構和內容的更多相關資訊，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哪些內容。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。具有該原則的使用者可以從 AWS Management Console AWS CLI、或 AWS API 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。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的建立 IAM 政策。

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。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶. 受管政策包括 AWS 受管政策和客戶管理的策略。若要了解如何在受管政策及內嵌政策間選擇，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。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些委託人 (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 和 Amazon VPC 是支援 ACL 的服務範例。 AWS WAF若要進一步了解 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原則類型。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。如需許可範圍的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 實體許可範圍。
- 服務控制策略（SCP）— SCP 是 JSON 策略，用於指定中組織或組織單位（OU）的最大權限。AWS Organizations AWS Organizations 是一種用於分組和集中管理您企業擁 AWS 帳戶 有的多個服務。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 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需組織和 SCP 的更多相關資訊，請參閱 AWS Organizations 使用者指南中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。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。如需更多資訊，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。要了解如何在涉及多個政策類型時 AWS 確定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

AWS 如何與 IAM HealthImaging 搭配使用

在您使用 IAM 管理存取權限之前 HealthImaging，請先了解哪些 IAM 功能可搭配使用 HealthImaging。

您可以搭配 AWS 使用的 IAM 功能 HealthImaging

IAM 功能	HealthImaging 支持
身分型政策	是
資源型政策	否
政策動作	是
政策資源	是
政策條件索引鍵 (服務特定)	是
ACL	否
ABAC(政策中的標籤)	部分
臨時憑證	是
主體許可	是
服務角色	是
服務連結角色	否

若要深入瞭解如何以 HealthImaging 及其他 AWS 服務如何使用大多數 IAM 功能，請參閱 IAM 使用者指南中的搭配 IAM 使用的AWS 服務。

以身分識別為基礎的原則 HealthImaging

支援身分型政策	是

身分型政策是可以連接到身分 (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。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

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。

以身分識別為基礎的原則範例 HealthImaging

若要檢視以 HealthImaging 身分為基礎的原則範例，請參閱。AWS 的身分型政策範例 HealthImaging

以資源為基礎的政策 HealthImaging

支援以資源基礎的政策	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。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

若要啟用跨帳戶存取，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，作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時 AWS 帳戶，受信任帳戶中的 IAM 管理員也必須授與主體實體 (使用者或角色) 權限，才能存取資源。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 角色與資源型政策有何差異。

的政策動作 HealthImaging

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哪些內容。也就是說，哪個主體在什麼条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。原則動作通常與關聯的 AWS API 作業具有相同的名稱。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 HealthImaging中 AWS 定義的動作。

中的策略動作在動作之前 HealthImaging 使用下列前置詞：

```
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"
         ]
```

若要檢視以 HealthImaging 身分為基礎的原則範例，請參閱。AWS 的身分型政策範例 HealthImaging

的政策資源 HealthImaging

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哪些內容。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 ARN 的清單，請參閱服務授權參考 HealthImaging中 AWS 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。

若要檢視以 HealthImaging 身分為基礎的原則範例，請參閱。AWS 的身分型政策範例 HealthImaging

的政策條件索引鍵 HealthImaging

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哪些內容。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，請使用邏輯OR運算來 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條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

若要查看 HealthImaging 條件金鑰清單，請參閱服務授權參考 HealthImaging中的 AWS 條件金鑰。若要了解可以使用條件金鑰的動作和資源，請參閱 AWS 定義的動作 HealthImaging。

若要檢視以 HealthImaging 身分為基礎的原則範例，請參閱。AWS 的身分型政策範例 HealthImaging

ACL 在 HealthImaging

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 與 HealthImaging

支持 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 根據人員的工作職能 (在角色之外稱 AWS 為) 來定義權限。如需詳細資訊，請參閱 IAM 使用者指南中的比較 ABAC 與傳統 RBAC 模型。

阿巴克與 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。任何具有`SearchImageSets`動作存取權的人都可以存取資料倉庫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，影像集必須具有與資料倉庫相同的標籤。如需詳細資訊，請參閱 使用 AWS 標記資源 HealthImaging。

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。在中 AWS，這些屬性稱為標籤。您可以將標籤附加到 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。

若要根據標籤控制存取，請使用 `aws:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 條件金鑰，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》中的什麼是 ABAC?。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)。

使用臨時登入資料 HealthImaging

支援臨時憑證	是

當您使用臨時憑據登錄時，某些 AWS 服務 不起作用。如需其他資訊，包括哪些 AWS 服務 與臨時登入資料搭配AWS 服務 使用，請參閱 IAM 使用者指南中的 IAM。

如果您使用除了使用者名稱和密碼以外的任何方法登入，則您正在 AWS Management Console 使用臨時認證。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取時，該程序會自動建立暫時認證。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》中的切換至角色 (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，而非使用長期存取金鑰。如需詳細資訊，請參閱 IAM 中的暫時性安全憑證。

的跨服務主體權限 HealthImaging

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，您會被視為主體。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 HealthImaging中適用於 AWS 的動作、資源和條件金鑰。

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。如需更多資訊，請參閱 IAM 使用者指南中的建立角色以委派許可給 AWS 服務。

> **Warning**
>
> 變更服務角色的權限可能會中斷 HealthImaging 功能。只有在 HealthImaging 提供指引時才編輯服務角色。

服務連結角色 HealthImaging

支援服務連結角色。	否

服務連結角色是一種連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。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 的身分型政策範例 HealthImaging

依預設，使用者和角色沒有建立或修改 HealthImaging資源的權限。他們也無法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 來執行工作。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。

若要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。

有關 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 ARN 格式，請參閱服務授權參考中 A AWS wesome 的動作、資源和條件鍵。

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

以身分識別為基礎的政策會決定某人是否可以建立、存取或刪除您帳戶中的 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 受管原則並邁向最低權限權限 — 若要開始授與使用者和工作負載的權限，請使用可授與許多常見使用案例權限的AWS 受管理原則。它們在您的 AWS 帳戶. 建議您透過定義特定於您使用案例的 AWS 客戶管理政策，進一步降低使用權限。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。

- 套用最低許可許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的權限。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。您也可以使用條件來授與對服務動作的存取權 (如透過特定 AWS 服務) 使用 AWS CloudFormation。如需更多資訊，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，可協助您編寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 IAM Access Analyzer 政策驗證。
- 需要多因素身份驗證 (MFA) — 如果您的案例需要 IAM 使用者或根使用者 AWS 帳戶，請開啟 MFA 以獲得額外的安全性。若要在呼叫 API 作業時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 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，您必須擁有最少一組許可。這些權限必須允許您列出和檢視有關 AWS 帳戶. HealthImaging 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。

您不需要為僅對 AWS CLI 或 AWS API 進行呼叫的使用者允許最低主控台權限。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。

若要確保使用者和角色仍可使用 HealthImaging 主控台，請同時將 HealthImaging *ConsoleAccess*或受*ReadOnly* AWS 管理的原則附加至實體。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。此原則包含在主控台上或以程式設計方式使用 AWS CLI 或 AWS API 完成此動作的權限。

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

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS適用於 AWS 的受管政策 HealthImaging

AWS 受管政策是由 AWS 建立和管理的獨立政策。AWS 受管政策的設計在於為許多常見使用案例提供許可，如此您就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與您特定使用案例的最低權限許可，因為它們可供所有 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。

您無法更改 AWS 受管政策中定義的許可。如果 AWS 更新 AWS 受管政策中定義的許可，更新會影響政策連接的所有主體身分 (使用者、群組和角色)。在推出新的 AWS 服務 或有新的 API 操作可供現有服務使用時，AWS 很可能會更新 AWS 受管政策。

如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html#aws-managed-policies中的 AWS 受管政策。

主題

- AWS受管理策略： AWSHealthImagingFullAccess
- AWS受管理策略： AWSHealthImagingReadOnlyAccess
- HealthImaging AWS受管理策略的更新

AWS受管理策略： AWSHealthImagingFullAccess

您可將 AWSHealthImagingFullAccess 政策連接到 IAM 身分。

此原則會授與所有 HealthImaging 動作的管理權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "medical-imaging:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "medical-imaging.amazonaws.com"
```

```
                }
            }
        }
    ]
}
```

AWS受管理策略： AWSHealthImagingReadOnlyAccess

您可將 AWSHealthImagingReadOnlyAccess 政策連接到 IAM 身分。

此政策授予特定 AWS HealthImaging 動作的唯讀權限。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "medical-imaging:GetDICOMImportJob",
            "medical-imaging:GetDatastore",
            "medical-imaging:GetImageFrame",
            "medical-imaging:GetImageSet",
            "medical-imaging:GetImageSetMetadata",
            "medical-imaging:ListDICOMImportJobs",
            "medical-imaging:ListDatastores",
            "medical-imaging:ListImageSetVersions",
            "medical-imaging:ListTagsForResource",
            "medical-imaging:SearchImageSets"
        ],
        "Resource": "*"
    }]
}
```

HealthImaging AWS受管理策略的更新

檢視 HealthImaging 自此服務開始追蹤這些變更以來的AWS受管理策略更新詳細資料。如需有關此頁面變更的自動警示，請訂閱「新聞稿」頁面上的 RSS 摘要。

變更	描述	日期
HealthImaging 開始追蹤變更	HealthImaging 開始追蹤其 AWS受管理策略的變更。	2023年7月19日

疑難排解 AWS HealthImaging 身分和存取

使用下列資訊可協助您診斷和修正使用和 IAM 時可能會遇到的 HealthImaging 常見問題。

主題

- 我沒有執行操作的授權 HealthImaging
- 我沒有授權執行 iam：PassRole
- 我想允許我以外的人訪 AWS 帳戶 問我的 HealthImaging 資源

我沒有執行操作的授權 HealthImaging

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 系統管理員。您的管理員提供您的簽署憑證。

我沒有授權執行 iam：PassRole

如果您收到未獲授權執行iam:PassRole動作的錯誤訊息，則必須更新您的原則以允許您將角色傳遞給 HealthImaging。

有些 AWS 服務 允許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 使用者marymajor嘗試使用主控台執行中的動作時，會發生下列範例錯誤 HealthImaging。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 系統管理員。您的管理員提供您的簽署憑證。

我想允許我以外的人訪 AWS 帳戶 問我的 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：

- 若要瞭解是否 HealthImaging 支援這些功能，請參閱AWS 如何與 IAM HealthImaging 搭配使用。
- 若要了解如何提供對您所擁有資源 AWS 帳戶 的存取權，請參閱 IAM 使用者指南中您擁有的另一 AWS 帳戶 個 IAM 使用者提供存取權限。
- 若要了解如何將資源存取權提供給第三方 AWS 帳戶，請參閱 IAM 使用者指南中的提供第三方 AWS 帳戶 擁有的存取權。
- 若要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在外部進行身分驗證的使用者 (聯合身分)。
- 若要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 角色與資源型政策的差異。

AWS 中的記錄和監控 HealthImaging

記錄和監控是維護 AWS 安全性、可靠性、可用性和效能的重要組成部分 HealthImaging。 AWS提供下列記錄與監控工具，供您觀看 HealthImaging、在發生錯誤時回報，並在適當時自動採取動作：

- AWS CloudTrail 擷取您 AWS 帳戶發出或代表發出的 API 呼叫和相關事件，並傳送記錄檔案至您指定的 Simple Storage Service (Amazon S3) 儲存貯體。您可以找出哪些使用者和帳戶呼叫 AWS、發出呼叫的來源 IP 地址，以及呼叫的發生時間。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南。

- Amazon 會即時 CloudWatch監控您的AWS資源和執行AWS的應用程式。您可以收集和追蹤指標、建立自訂儀表板，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 CloudWatch 追蹤 Amazon EC2 執行個體的 CPU 使用率或其他指標，並在需要時自動啟動新執行個體。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南。

主題

使用記錄 AWS HealthImaging API 呼叫 AWS CloudTrail

AWS HealthImaging 與服務整合 AWS CloudTrail，可提供中使用者、角色或服務所採取的動作記錄的 AWS 服務 HealthImaging。 CloudTrail 擷取 HealthImaging 作為事件的所有 API 呼叫。擷取的呼叫包括來自 HealthImaging 主控台的呼叫和 HealthImaging API 作業的程式碼呼叫。如果您建立追蹤，您可以開啟連續交付 CloudTrail事件至 Amazon S3 儲存貯體，包括 HealthImaging. 如果您未設定追蹤，您仍然可以在 [事件歷程記錄] 中檢視 CloudTrail 主控台中最近的事件。使用收集的資訊 CloudTrail，您可以判斷提出的要求 HealthImaging、提出要求的 IP 位址、提出要求的人員、提出要求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱使AWS CloudTrail 用者指南。

建立追蹤

CloudTrail 在您建立帳戶 AWS 帳戶 時為您開啟。當活動發生在中時 HealthImaging，該活動會與事件歷史記錄中的其他 AWS 服務 CloudTrail 事件一起記錄在事件中。您可以檢視、搜尋和下載您的 AWS 帳戶. 如需詳細資訊，請參閱使用 CloudTrail 事件歷程記錄檢視事件。

> **Note**
>
> 若要 HealthImaging 在中檢視 AWS 的 CloudTrail 事件歷史記錄 AWS Management Console，請轉至查詢屬性功能表，選取事件來源，然後選擇`medical-imaging.amazonaws.com`。

對於您的事件的持續記錄 AWS 帳戶，包括事件 HealthImaging，請創建一個跟踪。追蹤可 CloudTrail 將日誌檔交付到 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。追蹤記錄來自 AWS 分區中所有區域的事件，並將日誌檔傳送到您指定的 Amazon S3 儲存貯體。此外，您還可以設定其他 AWS 服務，以進一步分析 CloudTrail 記錄中收集的事件資料並採取行動。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務與整合
- 設定 Amazon SNS 通知 CloudTrail
- 從多個區域接收 CloudTrail 日誌文件並從多個帳戶接收 CloudTrail 日誌文件

> **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，記錄會套用至每個啟用它的 HealthImaging API 呼叫的管理事件。資料事件是可計費的，通常會保留給每秒交易量高 (tps) 的 API，因此您可以選擇退出具有成本目的的 CloudTrail 記錄。
> 使用時 HealthImaging，AWS API 參考中列出的所有 HealthImaging API 動作都被視為管理事件，但不包括 `GetImageFrame`. `GetImageFrame`動作會以資料事件的 CloudTrail 形式登入，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。

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：

- 要求是使用根使用者登入資料還是 AWS Identity and Access Management (IAM) 使用者登入資料提出。
- 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。

瞭解記錄項目

追蹤是一種組態，可讓事件以日誌檔的形式傳遞到您指定的 Amazon S3 儲存貯體。 CloudTrail 記錄檔包含一或多個記錄項目。事件代表來自任何來源的單一請求，包括有關請求的操作，動作的日期和時間，請求參數等信息。 CloudTrail 日誌文件不是公共 API 調用的有序堆棧跟踪，因此它們不會以任何特定順序顯示。

下列範例顯示示範GetDICOMImportJob動作的 CloudTrail 記錄項目。 HealthImaging


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "XXXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
        "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/
ce6d90ba-5fba-4456-a7bc-f9bc877597c3"
        "accountId": "123456789012",
        "accessKeyId": "XXXXXXXXXXXXXXXXXXXX",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "XXXXXXXXXXXXXXXXXXXXX",
                "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
                "accountId": "123456789012",
                "userName": "TestAccessRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-28T15:52:42Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-28T16:02:30Z",
    "eventSource": "medical-imaging.amazonaws.com",
    "eventName": "GetDICOMImportJob",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-
Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync
 http/Apache cfg/retry-mode/standard",
    "requestParameters": {
        "jobId": "5d08d05d6aab2a27922d6260926077d4",
        "datastoreId": "12345678901234567890123456789012"
    },
    "responseElements": null,
    "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
    "eventID": "26307f73-07f4-4276-b379-d362aa303b22",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
```

```
    "recipientAccountId": "824333766656",
    "eventCategory": "Management"
}
```

HealthImaging 使用亞馬遜監控 AWS CloudWatch

您可以 HealthImaging 使用來監控 AWS CloudWatch，這會收集原始資料，並將其處理為可讀且近乎即時的指標。這些統計數據保留 15 個月，因此您可以使用該歷史信息，並更好地了解 Web 應用程序或服務的性能。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**
>
> 會報告所有 HealthImaging API 的量度。

下表列出的測量結果和維度 HealthImaging。每個都會以使用者指定資料範圍的頻率計數呈現。

指標

指標	描述
呼叫計數	對 API 的呼叫次數。這可以針對帳戶或指定的資料倉庫報告。 單位：計數 有效統計資料：總和、計數 維度：作業、資料存放區 ID、資料存放區類型

您可以 HealthImaging 使用AWS Management Console、或 CloudWatch API 取AWS CLI得的指標。您可以透過其中 CloudWatch 一個 Amazon AWS 軟體開發套件 (開發套件) 或 CloudWatch API 工具使用 API。 HealthImaging 控制台根據來自 CloudWatch API 的原始數據顯示圖形。

您必須擁有適當的 CloudWatch 權限才能監 HealthImaging 視 CloudWatch。如需詳細資訊，請參閱《CloudWatch 使用指南》 CloudWatch中的〈身分識別與存取管理〉。

檢視 HealthImaging 量度

若要檢視量度 (CloudWatch 主控台)

1. 登入 AWS Management Console 並開啟 CloudWatch 主控台。
2. 選擇「度量」，選擇「所有量度」，然後選擇「AWS/ 醫學影像」。
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。

使用建立鬧鐘 CloudWatch

CloudWatch 警示會監視指定時段內的單一指標，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。動作或動作是根據您指定數個期間內，相對於指定臨界值的測量結果值。 CloudWatch 也可以在警示狀態變更時傳送 Amazon SNS 訊息給您。

CloudWatch 警報只有在狀態變更且您指定的期間內持續存在時，才會呼叫動作。如需詳細資訊，請參閱使用 CloudWatch 鬧鐘。

適用於 AWS 的合規驗證 HealthImaging

第三方稽核員會在多個合規計劃中評估 AWS HealthImaging 的安全性和合AWS規性。對於 HealthImaging，這包括 HIPAA。

如需特定合規計劃範圍內的 AWS 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 合規計畫。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱在 AWS Artifact 中下載報告。

您在使用 AWS 時的合規責任取決 HealthImaging 於資料的敏感度、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遵循法規：

- AWS合作夥伴解決方案 — 適用於安全性與合規性的自動化參考部署指南，討論架構考量事項，並提供在上部署以安全性和法規遵循為重點的 AWS
- HIPAA 安全與合規架構白皮書：本白皮書說明公司可如何運用 AWS 來建立 HIPAA 合規的應用程式。
- GxP 系統開啟 AWS — 本白皮書提供如何處理 GXP 相關AWS法規遵循和安全性的資訊，並提供在 GxP 環境中使用AWS服務的指引。

- AWS 合規資源 – 這組手冊和指南可能適用於您的產業和位置。
- 使用規則評估資源 — 評AWS Config估您的資源配置是否符合內部實踐，行業準則和法規。
- AWS Security Hub – 此 AWS 服務可供您檢視 AWS 中的安全狀態，可助您檢查是否符合安全產業標準和最佳實務。

AWS 中的彈性 HealthImaging

AWS 全球基礎架構是以 AWS 區域 與可用區域為中心建置的。AWS 區域 提供多個分開且隔離的實際可用區域，並以具備低延遲、高輸送量和高度備援特性的聯網相互連結。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、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域 與可用區域的詳細資訊，請參閱AWS全球基礎架構。

除了AWS全球基礎設施之外，AWS 還 HealthImaging 提供多種功能來協助支援您的資料彈性和備份需求。

AWS 中的基礎設施安全 HealthImaging

AWS HealthImaging 是受管服務，受 Amazon Web Services：安AWS全程序概觀白皮書中所述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 HealthImaging 過網路存取。用戶端必須支援傳輸層安全性 (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) 來產生用來簽署要求的臨時安全登入資料。

使用建立 AWS HealthImaging 資源 AWS CloudFormation

AWS 整合了 HealthImaging 這項服務AWS CloudFormation，可協助您建立資源模型和設定資AWS源，以減少建立和管理資源和基礎設施的時間。您可以建立一個範本，以描述所需的所有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。只需描述一次您的資源，即可在多個 AWS 帳戶 帳戶與區域內重複佈建相同資源。

HealthImaging 和AWS CloudFormation範本

若要佈建和設定 HealthImaging 與相關服務的資源，您必須瞭解AWS CloudFormation範本。範本是以 JSON 或 YAML 格式化的文本檔案。而您亦可以透過這些範本的說明，了解欲在 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 Designer 協助您開始使用 AWS CloudFormation 範本。如需詳細資訊，請參閱 AWS CloudFormation 使用者指南中的什麼是 AWS CloudFormation Designer？。

AWS HealthImaging 支援使用建立資料存放區AWS CloudFormation。如需詳細資訊，包括佈建資 HealthImaging 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》https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html
- AWS CloudFormation API 參考
- 《AWS CloudFormation 命令列介面使用者指南》https://docs.aws.amazon.com/cloudformation-cli/latest/userguide/what-is-cloudformation-cli.html

AWS HealthImaging 和介面 VPC 端端點 ()AWS PrivateLink

您可以在 VPC 和 AWS HealthImaging 建立介面 VPC 端點之間建立私人連線。界面端點採用這種技術 AWS PrivateLink，您可以使用這種技術在沒有網際網路閘道、NAT 裝置、VPN 連線或 AWS Direct Connect 的情況下私有存取 HealthImaging API。VPC 中的執行個體不需要公有 IP 位址即可與 HealthImaging API 通訊。您的 VPC 和 HealthImaging 不離開 Amazon 網絡之間的流量。

每個介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 使用者指南中的介面虛擬私人雲端端點 (AWS PrivateLink)。

主題

- HealthImaging VPC 端點的考量
- 為建立介面 VPC 端點 HealthImaging
- 建立 VPC 端點原則 HealthImaging

HealthImaging VPC 端點的考量

在為其設定介面 VPC 端點之前 HealthImaging，請務必先檢閱 Amazon VPC 使用者指南中的介面端點屬性和限制。

HealthImaging 支援從您的 VPC 呼叫所有 AWS HealthImaging 動作。

為建立介面 VPC 端點 HealthImaging

您可以使用 Amazon VPC 主控台或 AWS Command Line Interface ()AWS CLI建立 HealthImaging 服務的 VPC 端點。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的建立介面端點。

為 HealthImaging 使用下列服務名稱建立 VPC 端點：

- COM. 亞馬遜。 ##. #學影像
- COM. 亞馬遜。 ##。 runtime-medical-imaging
- COM. 亞馬遜。 ##。 dicom-medical-imaging

> **Note**
>
> 必須啟用私有 DNS 才能使用 PrivateLink。

您可以 HealthImaging 使用該區域的預設 DNS 名稱發出 API 要求，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。

建立 VPC 端點原則 HealthImaging

您可以將端點策略附加到控制對其存取的 VPC 端點。 HealthImaging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：用於動作的 VPC 端點原則 HealthImaging

以下是的端點策略範例 HealthImaging。連接至端點時，此策略會授與所有資源上所有主參與者 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\":\"*\"}]}"
```

跨帳戶匯入 AWS HealthImaging

透過跨帳戶/跨區域匯入，您可以將資料從位於其他支援區域的 Amazon S3 儲存貯體匯入 HealthImaging 資料存放區。您可以跨 AWS 帳戶、其他 Organ AWS izations 擁有的帳戶以及開放資料來源 (例如位於開放資料登錄中的影像資料共享資料 (IDC) 匯入資料。 AWS

HealthImaging 跨帳戶/跨區域匯入使用案例包括：

- 醫療影像 SaaS 產品從客戶帳戶匯入 DICOM 資料
- 從多個 Amazon S3 輸入儲存貯體填入一個 HealthImaging 資料存放區的大型組織

- 研究人員在多機構臨床研究中安全共享數據

若要使用跨帳戶匯入

1. Amazon S3 輸入 (來源) 儲存貯體擁有者必須授與資 HealthImaging 料存放區擁有者`s3:ListBucket`和`s3:GetObject`許可。
2. 資 HealthImaging 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. 資 HealthImaging 料存放區擁有者在啟動匯入任務時，必須`inputOwnerAccountId`為 Amazon S3 輸入儲存貯體提供。

> **Note**
>
> 透過提供`inputOwnerAccountId`，資料存放區擁有者可驗證輸入的 Amazon S3 儲存貯體屬於指定帳戶，以維持符合業界標準並降低潛在的安全風險。

下列`startDICOMImportJob`程式碼範例包含選用`inputOwnerAccountId`參數，可套用至開始匯入工作區段中的所有 AWS CLI 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 inputOwnerAccountId) {

    try {
        StartDicomImportJobRequest startDicomImportJobRequest =
 StartDicomImportJobRequest.builder()
                .jobName(jobName)
                .datastoreId(datastoreId)
                .dataAccessRoleArn(dataAccessRoleArn)
                .inputS3Uri(inputS3Uri)
                .outputS3Uri(outputS3Uri)
                .inputOwnerAccountId(inputOwnerAccountId)
```

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

    return "";
}
```

AWS HealthImaging 參考資料

下列參考資料適用於 AWS HealthImaging。

> **Note**
>
> 所有 HealthImaging 動作和資料類型都位於單獨的參考中。如需詳細資訊，請參閱 AWS HealthImaging API 參考。

主題

AWS 的 DICOM 支援 HealthImaging

AWS HealthImaging 支援特定的 DICOM 元素和傳輸語法。熟悉受支援的「患者」、「研究」和「系列」層級 DICOM 資料元素，因為 HealthImaging 中繼資料金鑰是以這些元素為基礎。開始匯入之前，請確認您的醫療影像資料符合支援 HealthImaging的傳輸語法和 DICOM 元素限制。

> **Note**
>
> AWS 目前 HealthImaging 不支援二進位分段影像或圖示影像序列像素資料。

主題

- DICOM 元素限制
- DICOM 中繼資料限制

支援的 SOP 類別

使用 AWS HealthImaging，您可以匯入使用任何 SOP 類別 UID 編碼的 DICOM P10 服務物件配對 (SOP) 執行個體，包括淘汰和私有。所有私有屬性也會保留下來。

元数据规范化

將 DICOM P10 資料匯入 AWS 時 HealthImaging，資料會轉換為由中繼資料和影像畫面 (像素資料) 組成的映像集。在轉換過程 HealthImaging 中，會根據特定版本的 DICOM 標準產生中繼資料金鑰。HealthImaging 目前根據 DICOM PS3.6 2022 b 資料字典產生並支援中繼資料金鑰。

AWS 在「患者」、「研究」和「系列」層級 HealthImaging 支援下列 DICOM 資料元素。

病人水平元素

> **Note**
>
> 如需每個病患等級元素的詳細說明，請參閱 DICOM 資料元素登錄。

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
```

研究水平元素

> **Note**
>
> 如需每個研究層級元素的詳細說明，請參閱 DICOM 資料元素登錄。

AWS HealthImaging 支援下列學習等級元素：

```
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
```

系列級元素

> **Note**
>
> 如需每個「序列」層次元素的詳細說明，請參閱 DICOM 資料元素登錄。

AWS HealthImaging 支援下列系列層級元素：

```
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 匯入使用下表中的傳輸語法編碼的 DICOM P10 SOP 執行個體。除了儲存 SOP 執行個體之外，針對使用下列傳輸語法編 HealthImaging 碼的 SOP 執行個體，將影像框 (像素資料) 轉碼為 HTJ2K：

轉移語法 UID	傳輸語法名稱
1.2.840.10008.1.2	隱含虛擬實境端序：DICOM 的預設傳輸語法
1.2.840.10008.1.2.1	露骨 VR 小端序
1.2.840.10008.1.1.99	放氣顯式 VR 小端序
1.2.840.10008.1.2.2	露骨 VR 大端序
1.2.840.10008.1.2.4.50	JPEG 基線（處理 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 基線（程序 2 和 4）：失真 JPEG 12 位元影像壓縮的預設傳輸語法（僅限處理 4）
1.2.840.10008.1.2.4.57	JPEG 不失真非階層式 (處理 14)
1.2.840.10008.1.2.4.70	JPEG 無失真、非階層式、一階預測 (程序 14 [選取值 1])：不失真 JPEG 影像壓縮的預設傳輸語法
1.2.840.10008.1.2.4.80	JPEG-LS 無損圖像壓縮
1.2.840.10008.1.2.4.81	JPEG-LS 有損（近乎無損）圖像壓縮
1.2.840.10008.1.2.4.90	影像壓縮 (僅適用於不失真)
1.2.840.10008.1.2.4.91	影像壓縮
1.2.840.10008.1.4.201	高傳輸量的影像壓縮 (僅適用於無損)

轉移語法 UID	傳輸語法名稱
1.2.840.10008.1.4.202	採用 RPCL 選項影像壓縮功能的高輸送量 JPEG 2000 (僅限不失真)
1.2.840.10008.1.4.203	高通量圖像壓縮
1.2.840.10008.1.2.5	RLE 無失真保護

DICOM 元素限制

當您匯入資料和更新映像集中繼資料屬性時，AWS 會 HealthImaging 套用 DICOM 元素限制。下面列出了匯入和中繼資料更新條件約束。

將醫療影像資料匯入 AWS 時 HealthImaging，下列 DICOM 元素會套用最大長度限制。若要成功匯入，請確定您的資料未超過最大長度限制。

DICOM 匯入限制

HealthImaging 關鍵字	關鍵字	迪克密鑰	長度限制
二氧化合物	耐心病	(0010,0020)	最小值：0，最大：64
狄克 PatientName	病人姓名	(0010,0010)	最小值：0，最大值：256
狄克 PatientBirthDate	病人 BirthDate	(0010,0030)	最小值:0, 最大:18
狄克 PatientSex	PatientSex	(0010,0040)	最小 0 分鐘，最大：16
二元 StudyInstance	StudyInstanceUID	(002 萬)	最小值：0，最大：64
狄克 StudyId	研究	(0020,0010)	最小 0 分鐘，最大：16
狄克 StudyDescription	StudyDescription	(0008,1030)	最小值：0，最大：64
狄克 NumberOfStudy RelatedSeries	数量相关系列	(0020,1206)	最小時間：0，最大

HealthImaging 關鍵字	關鍵字	迪克密鑰	長度限制
狄克 NumberOfStudy RelatedInstances	NumberOfStudyRelated實例	(0020,1208)	最小時間：0，最大
狄克 Accession Number	AccessionNumber	(0008,0050)	最小 0 分鐘，最大：16
狄克 StudyDate	StudyDate	(0008,0020)	最小值:0, 最大:18
狄克 StudyTime	StudyTime	(0008,0030)	最小 0 分鐘，最大：28

DICOM 中繼資料限制

當您使用`UpdateImageSetMetadata`更新 HealthImaging 中繼資料屬性時，會套用下列 DICOM 條件約束。

- 無法更新或移除患者/研究/序列/執行處理層次屬性的非公開屬性，除非更新限制同時適用於和 `updatableAttributes` `removableAttributes`
- 無法更新下列 AWS HealthImaging 產生的屬性：`SchemaVersion`、`DatastoreID`、`ImageSetID`、`PixelData`、、`Checksum`、`Width`、`Height`、、`M`
- 無法更新下列 DICOM 屬性：`Tag.PixelData`、`Tag.StudyInstanceUID`、、`Tag.SeriesInstanceUID`、`Tag.SOPInstance` `Tag.StudyID`
- 無法使用 VR 類型 SQ (巢狀屬性) 更新屬性
- 無法更新多值屬性
- 無法使用與屬性 VR 類型不相容的值更新屬性
- 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，如果在客戶有效負載請求中的「研究」級別給出了「患者」級別屬性，則該請求可能會失效。
- 如果關聯的屬性模組不存在於現有模組中，則無法更新屬性`ImageSetMetadata`。例如，`seriesInstanceUID`如果具有的序列不存在於現有影像集中繼資料中，`seriesInstanceUID`就不允許更新的屬性。

AWS HealthImaging 像素資料驗證

AWS HealthImaging 透過檢查每個影像的無失真編碼和解碼狀態，提供內建的像素資料驗證。

- 當匯入工作在匯入影像之前擷取影像的原始像素品質狀態時，影像上線程序便會開始。使用 CRC32 演算法為每個影像產生唯一的不可變影像框解析度檢查碼 (IFRC)。
- IFRC 是根據每個圖像幀的分辨率級別計算的。總和檢查碼值會出現在中繼資料文件中的清單中，從基準解析度到完整解析度排序。
- 匯入影像之後，會立即解碼影像，並計算新的 IFRC。 HealthImaging將原始影像的完整解析度 IFRC 與匯入影像的新 IFRC 進行比較，以確認正確性。
- 匯入工作輸出記錄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

1. 匯入醫療影像資料後，檢視匯入工作輸出記錄中擷取的每個影像集描述性成功 (或錯誤條件)。`job-output-manifest.json`如需有關 `job-output-manifest.json` 的詳細資訊，請參閱 瞭解匯入工作。
2. 使用`GetImageSetMetadata`動作擷取影像集的相關中繼資料。如需詳細資訊，請參閱取得影像集中繼資料。

   影像集的中繼資料包含有關影像框 (像素資料) 的資訊。`PixelDataChecksumFromBaseToFullResolution`包含每個解析度等級的 IFRC (總和檢查碼)。以下是作為匯入工作程序一部分所產生之 IFRC 的中繼資料輸出範例。

```
"ImageFrames": [{
  "ID": "678906789067890123451234512345123451234512",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 128,
      "Height": 128,
      "Checksum": 2928338830
    },
    {
      "Width": 256,
      "Height": 256,
      "Checksum": 1362274918
    },
    {
      "Width": 512,
```

```
      "Height": 512,
      "Checksum": 2510355201
    }
]
```

3. 要驗證像素數據，請訪問上的 Pixel 數據驗證過程， GitHub 並按照README.md文件中的說明獨立驗證由所使用的各種HTJ2K 解碼程式庫圖像的無損圖像處理。 HealthImaging由於資料會依解析度等級逐步載入，因此您可以在最後計算 IFRC 以取得原始輸入資料，並將其與相同解析度的中 HealthImaging 繼資料中提供的 IFRC 值進行比較，以驗證像素資料。

適用於 AWS 的 HTJ2K 解碼程式庫 HealthImaging

在匯入期間，AWS 會以高輸送量 JPEG 2000 (HTJ2K) 無失真格式對所有影像畫面 (像素資料) 進行 HealthImaging 編碼，以提供一致的快速影像顯示和 HTJ2K 進階功能的普遍存取。由於影像框會在匯入期間以 HTJ2K 編碼，因此必須先將其解碼，然後才能在影像檢視器中進行檢視。

> **Note**
>
> HTJ2K 的定義在 JPEG2000 標準第十五部分。HTJ2K 保留了 JPEG2000 的先進功能，例如分辨率可擴展性，區域，並排，高位元深度，多通道和色彩空間支持。

主題

- 解碼程式庫
- 影像檢視器

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框架。

- NVIDIA 處理器 — 商用、GPU 加速
- 卡卡杜軟件-商業，C ++ 與 Java 和 .NET 綁定
- 開放原始碼 — 開放原始碼、C++ 和 WASM
- 開源 — 開放原始碼、C ++、爪哇
- 開放式 — 開源, Python
- 打開 JPEG — 開源, Python

影像檢視器

您可以在解碼影像框之後檢視影像框。AWS HealthImaging API 動作支援各種開放原始碼影像檢視器，包括：

- 開放 Health 影像基金會
- Cornerstone.js

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

服務端點是將主機和連接埠識別為 Web 服務進入點的 URL。每個 Web 服務請求都會包含一個端點。大多數 AWS 服務會為特定區域提供端點，以加快連線速度。下表列出 AWS 的服務端點 HealthImaging。

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 請求呼叫 AWS HealthImaging 動作，則必須根據呼叫的動作使用不同的端點。下列功能表列出 HTTP 要求的可用服務端點及其支援的動作。

HTTP 要求支援的 API 動作

data store, import, tagging

可透過端點存取下列資料存放區、匯入和標記動作：

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores
- DeleteDatastore
- 開始迪克 ImportJob
- GetDicom ImportJob
- 列表迪科 ImportJobs
- 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 提供了一個表示的二通網檢索 WADO-RS 服務。如需詳細資訊，請參閱 取得 DICOM 執行個體。

下列 DicomWeb 服務可透過端點存取：

```
https://dicom-medical-imaging.region.amazonaws.com
```

- 得到迪科姆

Service Quotas

服務配額定義為帳戶中資源、動作和項 AWS 目的最大值。

> **Note**
>
> 對於可調配額，您可以使用 Service Quotas 控制台申請增加配額。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 的預設配額 HealthImaging。

名稱	預設	可調整	描述
每個資料存放區的並行 CopyImageSet 要求數	每個受支援的區域：100	是	目前區域中每個資料存放 AWS 區的並行 CopyImageSet 要求上限
每個資料存放區的並行 DeleteImageSet 要求數	每個受支援的區域：100	是	目前區域中每個資料存放 AWS 區的並行 DeleteImageSet 要求上限
每個資料存放區的並行 UpdateImageSetMetadata 要求數	每個受支援的區域：100	是	目前區域中每個資料存放 AWS 區的並行 UpdateImageSetMetadata 要求上限
每個資料存放區並行匯入工作上限	ap-southeast-2:20 每個其他支援的區域：100 個	是	目前區域中每個資料儲存 AWS 區可同時匯入工作的最大數目
最大數據存儲	每個受支援的區域：10	是	目前區域中作用中資料存放 AWS 區的數目上限
每個 CopyImageSet 請求 ImageFrames 允許複製的最大數量	每個受支援的區域：1,000	是	當前 AWS 區域中每個 CopyImageSet 請求 ImageFrames 允許複製的最大數量

名稱	預設	可調整	描述
DICOM 匯入工作中的檔案數目上限	每個受支援的區域：5,000	是	目前區域中 DICOM 匯入工作中的檔案數目 AWS 上限
DICOM 匯入工作中巢狀資料夾的最大數目	每個受支援的區域：10,000	否	目前區域中 DICOM 匯入工作中巢狀資料夾的最大數目 AWS
接受的最大承載大小限制 (KB) UpdateImageSetMetadata	每個支持的地區：10 千字節	是	目前 AWS 區域中接受的最大承載大小限制 (以 KB 為單 UpdateImageSetMetadata 位)
DICOM 匯入工作中所有檔案的大小上限 (GB)	每個支持的地區：10 GB	否	目 AWS 前區域中 DICOM 匯入工作中所有檔案的大小上限 (GB)
DICOM 匯入工作中每個 DICOM P10 檔案的大小上限 (以 GB 為單位)	每個支持的地區：4 GB	否	目前區域中 DICOM 匯入工作中每個 DICOM P10 檔案的大小上限 (以 GB 為單位) AWS
ImageSetMetadata 每次匯入、複製和的最大大小限制 (MB) UpdateImageSet	每個支援的地區：50 MB	是	ImageSetMetadata 每個匯入、複製和 UpdateImageSet 目前 AWS 區域中的最大大小限制 (MB)

AWS HealthImaging 節流限制

您的 AWS 帳戶有適用於 AWS HealthImaging API 動作的節流限制。對於所有動作，如果超過節流限制，就會擲回`ThrottlingException`錯誤。如需詳細資訊，請參閱 AWS HealthImaging API 參考。

> **Note**
>
> 所有 HealthImaging API 動作的節流限制均可調整。若要請求調整節流限制，請聯絡 Sup AWS port 中心。

下表列出 AWS HealthImaging API 動作的節流限制。

AWS HealthImaging 節流限制

動作	節流率	油門爆裂
CreateDatastore	每秒 0.085	每秒 1 個
GetDatastore	10 tps	每秒 20 台
ListDatastores	每秒 5 茶	10 tps
DeleteDatastore	每秒 0.085	每秒 1 個
起始 ImportJob	每秒 0.25 美元	每秒 1 個
GetDicom ImportJob	每秒 25 個	每秒 50
列表 ImportJobs	10 tps	每秒 20 台
SearchImageSets	每秒 25 個	每秒 50
GetImageSet	每秒 25 個	每秒 50
GetImageSetMetadata	每秒 50	每秒
GetImageFrame	千	二千
獲取二科分 *	每秒 50	每秒
ListImageSetVersions	每秒 25 個	每秒 50
UpdateImageSetMetadata	每秒 0.25 美元	每秒 1 個
CopyImageSet	每秒 0.25 美元	每秒 1 個

動作	節流率	油門爆裂
DeleteImageSet	每秒 0.25 美元	每秒 1 個
TagResource	10 tps	每秒 20 台
ListTagsForResource	10 tps	每秒 20 台
UntagResource	10 tps	每秒 20 台

* 代表一個互聯網服務

AWS HealthImaging 範例專案

AWS 在上 HealthImaging 提供以下範例專案 GitHub。

[DICOM 擷取從現場部署到 AWS HealthImaging]

用於部署 IoT 邊緣解決方案的AWS無伺服器專案，該解決方案會接收來自 DICOM DIMSE 來源 (PACS、VNA、CT 掃描器) 的 DICOM 檔案，並將它們存放在安全的 Amazon S3 儲存貯體中。此解決方案會將 DICOM 檔案編入索引，並將要匯入 AWS 的每個 DICOM 系列排入佇列。HealthImaging它包含在由管理的邊緣執行的元件 AWS IoT Greengrass，以及在 Cloud 中AWS執行的 DICOM 擷取管線。

[瓷磚等級標記 (TLM) 代理]

HealthImaging 透過使用瓷磚層級標記 (TLM) 從 AWS 擷取影像畫面的AWS Cloud Development Kit (AWS CDK)專案，這是高輸送量 JPEG 2000 (HTJ2K) 的一項功能。如此一來，較低解析度影像的擷取時間就會更快。潛在的工作流程包括產生縮圖和漸進式載入影像。

[亞馬遜 CloudFront 交付]

一種AWS無伺服器專案，用於使用 HTTPS 端點建立 Amazon CloudFront 分發，該端點可快取 (使用 GET) 並從邊緣傳遞影像框架。根據預設，端點會使用 Amazon Cognito JSON 網頁權杖 (JWT) 驗證請求。身份驗證和請求簽署都是在邊緣使用 Lambda @Edge 完成的。這項服務是 Amazon 的一項功能，可 CloudFront 讓您在更接近應用程式使用者的位置執行程式碼，進而提升效能並減少延遲。沒有基礎設施需要管理。

AWS HealthImaging 檢視器 UI

用於部署具有後端身份驗證的前端 UI 的AWS Amplify專案，您可以 HealthImaging 使用漸進式解碼來檢視存放在 AWS 中的影像集中繼資料屬性和映像框 (像素資料)。您可以選擇性地整合上面的瓷磚層級標記 (TLM) 代理和/或 Amazon CloudFront 交付專案，以使用替代方法載入影像框。

若要檢視其他範例專案，請參閱上的 AWS HealthImaging 範例 GitHub。

搭 HealthImaging 配 AWS SDK 使用

AWS 軟件開發套件（SDK）可用於許多流行的編程語言。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
AWS SDK for C++	AWS SDK for C++ 程式碼範例
AWS CLI	AWS CLI 程式碼範例
AWS SDK for Go	AWS SDK for Go 程式碼範例
AWS SDK for Java	AWS SDK for Java 程式碼範例
AWS SDK for JavaScript	AWS SDK for JavaScript 程式碼範例
適用於 Kotlin 的 AWS SDK	適用於 Kotlin 的 AWS SDK 程式碼範例
AWS SDK for .NET	AWS SDK for .NET 程式碼範例
AWS SDK for PHP	AWS SDK for PHP 程式碼範例
AWS Tools for PowerShell	PowerShell 程式碼範例工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 程式碼範例
AWS SDK for Ruby	AWS SDK for Ruby 程式碼範例
適用於 Rust 的 AWS SDK	適用於 Rust 的 AWS SDK 程式碼範例
適用於 SAP ABAP 的 AWS SDK	適用於 SAP ABAP 的 AWS SDK 程式碼範例

SDK 文件	代碼範例
適用於 Swift 的 AWS SDK	適用於 Swift 的 AWS SDK 程式碼範例

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

下表顯示針對 AWS HealthImaging 服務和文件發行功能和更新的時間。

變更	描述	日期
GetDICOMInstance 用於返回 DICOM 數據	HealthImaging 提GetDICOMInstance 供傳回給定影像集之 DICOM 第 10 部資料 (.dcm檔案) 的服務。如需詳細資訊，請參閱 取得 DICOM 執行個體。	2024年5月15日
跨帳戶匯入	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 跨帳戶匯入 AWS HealthImaging。	2024年5月15日
影像集的搜尋增強功能	HealthImaging SearchImageSets 動作支援下列搜尋增強功能。如需詳細資訊，請參閱 搜尋影像集。 • 其他支援搜尋UpdatedAt 和 SeriesInstanceUID • 在開始時間和結束時間之間搜索 • 依Ascending 或排序搜尋結果 Descending • 在回應中傳回 DICOM 系列參數	2024年4月3日
匯入的檔案大小上限增加	HealthImaging 在匯入工作中，支援每個 DICOM P10 檔案的最大檔案大小為 4 GB。如	2024年3月6日

	需詳細資訊，請參閱 Service Quotas。	
轉換不失真和 HTJ2K 的語法	HealthImaging 支援工作匯入的下列傳輸語法。如需詳細資訊，請參閱 支援的傳輸語法。 • 1.2.840.10008.1.2.4.57 — JPEG 無損失真的非階層式 (處理十四) • 高傳輸量影像壓縮 (僅限無損) • 1.2.840.10008.1.4.202 — 採用 RPCL 選項影像壓縮功能的高輸送量 JPEG 2000 (僅限不失真) • 高通量圖像壓縮	2024年2月16日
測試的代碼示例	HealthImaging 文件提供了經過測試的程式碼範例，以 AWS CLI 及適用於 Python JavaScript、Java 和 C++ 的 AWS 開發套件。如需詳細資訊，請參閱 HealthImaging 使用 AWS SDK 的程式碼範例。	2023 年 12 月 19 日
匯入的最大檔案數目增加	HealthImaging 單一匯入工作最多可支援 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 單一匯入工作最多可支援 10,000 個巢狀資料夾。如需詳細資訊，請參閱 Service Quotas。	2023 年 12 月 1 日

更快的進口	HealthImaging 在所有支援的地區提供 20 倍的匯入速度。如需詳細資訊，請參閱 服務端點。	2023 年 12 月 1 日
CloudFormation 支持	HealthImaging 支援用於佈建資料存放區的基礎結構即程式碼 (IaC)。如需詳細資訊，請參閱 使用建立 AWS HealthImaging 資源 AWS CloudFormation。	2023 年 9 月 21 日
一般可用性	AWS 適用 HealthImaging 於美國東部 (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。	2023 年 7 月 26 日

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。