aws

Guida per gli sviluppatori

AWS HealthImaging

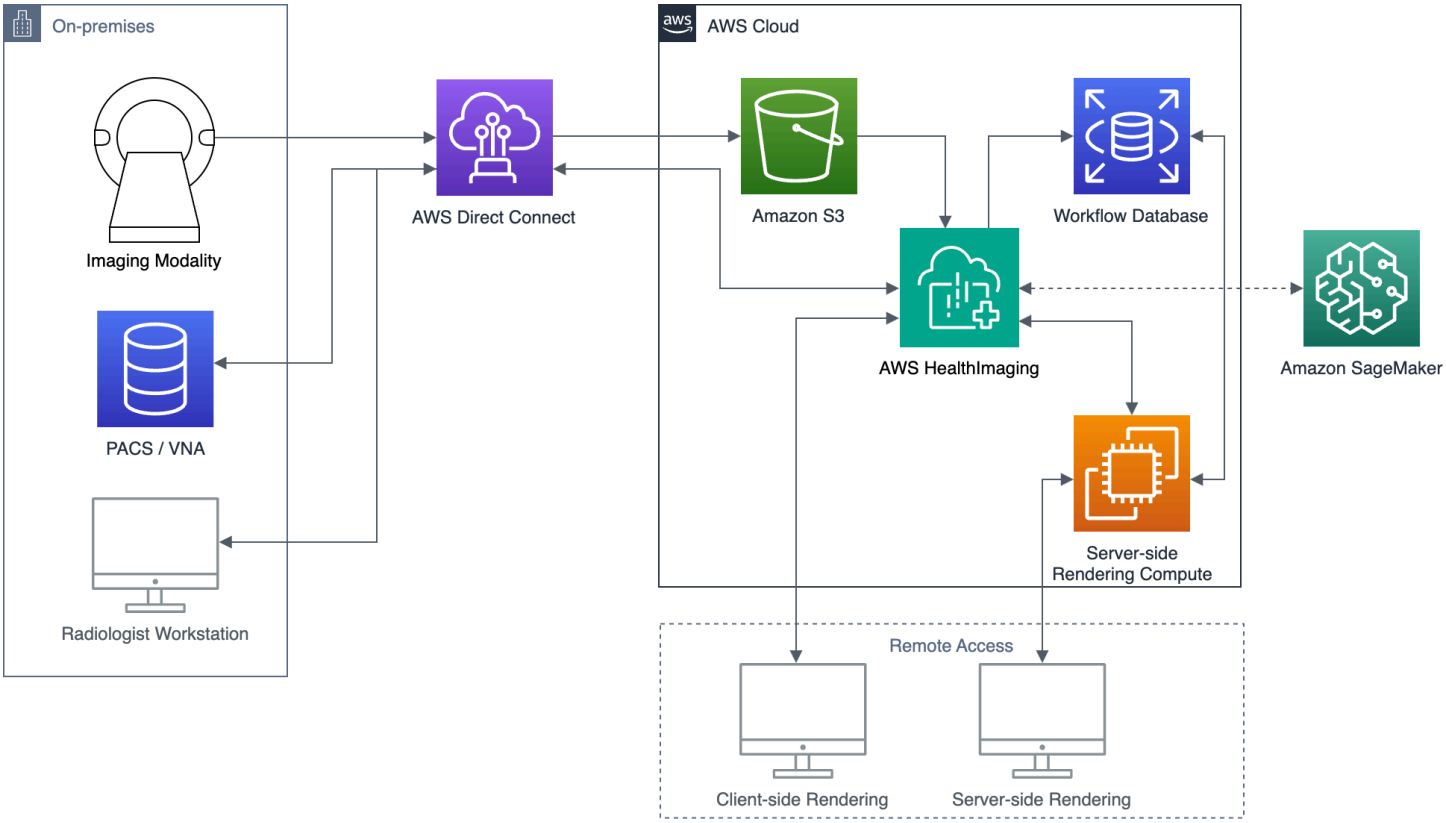
AWS HealthImaging: Guida per gli sviluppatori

Table of Contents

Che cos'è AWS HealthImaging?

AWS HealthImaging è un servizio idoneo all'HIPAA che consente agli operatori sanitari, alle organizzazioni del settore delle scienze biologiche e ai loro partner software di archiviare, analizzare e condividere immagini mediche nel cloud su scala petabyte. HealthImagingI casi d'uso includono:

- Imaging aziendale: archivia e trasmetti in streaming i dati di imaging medico direttamente dal AWS cloud, preservando al contempo prestazioni a bassa latenza e alta disponibilità.
- Archiviazione delle immagini a lungo termine: consente di risparmiare sui costi di archiviazione delle immagini a lungo termine mantenendo al contempo l'accesso al recupero delle immagini in meno di un secondo.
- Sviluppo AI/ML: esegui l'inferenza di intelligenza artificiale e apprendimento automatico (AI/ML) sul tuo archivio di immagini con il supporto di altri strumenti e servizi.
- Analisi multimodale: combina i dati di imaging clinico con AWS HealthLake (dati sanitari) e AWS HealthOmics (dati omici) per fornire informazioni utili alla medicina di precisione.



AWS HealthImaging fornisce l'accesso ai dati delle immagini (ad esempio X-Ray, TC, RM, Ultrasuoni) in modo che le applicazioni di imaging medico integrate nel cloud possano raggiungere prestazioni precedentemente possibili solo in locale. Con HealthImaging, riduci i costi di infrastruttura eseguendo le tue applicazioni di imaging medico su larga scala a partire da un'unica copia autorevole di ogni immagine medica. Cloud AWS

Argomenti

Avviso importante

HealthImaging AWS non sostituisce la consulenza, la diagnosi o il trattamento medico professionale e non è destinato a curare, trattare, mitigare, prevenire o diagnosticare alcuna malattia o condizione di salute. Sei responsabile dell'istituzione della revisione umana nell'ambito di qualsiasi utilizzo di AWS HealthImaging, anche in associazione a qualsiasi prodotto di terze parti destinato a informare il processo decisionale clinico. AWS HealthImaging deve essere utilizzato nell'assistenza ai pazienti o in scenari clinici solo dopo la revisione da parte di professionisti medici qualificati che applicano solide capacità di giudizio medico.

Caratteristiche di AWS HealthImaging

AWS HealthImaging offre le seguenti funzionalità.

Metadati DICOM intuitivi per gli sviluppatori

AWS HealthImaging semplifica lo sviluppo di applicazioni restituendo i metadati DICOM in un formato adatto agli sviluppatori. Dopo aver importato i dati di imaging, i singoli attributi dei metadati sono accessibili utilizzando parole chiave intuitive anziché numeri esadecimali di gruppo/elemento sconosciuti. Gli elementi DICOM a livello di paziente, studio e serie sono normalizzati, eliminando così la necessità per gli sviluppatori di applicazioni di gestire le incongruenze tra le

istanze SOP. Inoltre, i valori degli attributi dei metadati sono direttamente accessibili nei tipi di runtime nativi.

Decodifica delle immagini con accelerazione SIMD

AWS HealthImaging restituisce frame di immagini (dati pixel) codificati con High Throughput JPEG 2000 (HTJ2K), un codec di compressione delle immagini avanzato. HTJ2K sfrutta i vantaggi dei dati multipli a istruzione singola (SIMD) sui processori moderni per offrire nuovi livelli di prestazioni. HTJ2K è un ordine di grandezza più veloce di JPEG2000 e almeno due volte più veloce di tutte le altre sintassi di trasferimento DICOM. È possibile utilizzare WASM-SIMD per portare questa velocità estrema a visualizzatori Web a ingombro zero.

Verifica dei dati relativi ai pixel

AWS HealthImaging offre una verifica integrata dei dati dei pixel controllando lo stato di codifica e decodifica senza perdite di ogni immagine durante l'importazione. Per ulteriori informazioni, consulta Verifica dei dati relativi ai pixel.

Prestazioni leader del settore

AWS HealthImaging stabilisce un nuovo standard per le prestazioni di caricamento delle immagini grazie alla sua efficiente codifica dei metadati, alla compressione senza perdita di dati e all'accesso ai dati a risoluzione progressiva. L'efficiente codifica dei metadati consente ai visualizzatori di immagini e agli algoritmi di intelligenza artificiale di comprendere i contenuti di uno studio DICOM senza dover caricare i dati dell'immagine. Le immagini vengono caricate più velocemente senza alcun compromesso in termini di qualità dell'immagine grazie alla compressione avanzata delle immagini. La risoluzione progressiva consente un caricamento ancora più rapido delle immagini per miniature, aree di interesse e dispositivi mobili a bassa risoluzione.

Importazioni DICOM scalabili

HealthImaging Le importazioni da AWS sfruttano le moderne tecnologie native del cloud per importare più studi DICOM in parallelo. Gli archivi storici possono essere importati rapidamente senza influire sui carichi di lavoro clinici per i nuovi dati. Per informazioni sulle istanze SOP supportate e sulle sintassi di trasferimento, consulta. Supporto DICOM

Servizi correlati AWS

AWS HealthImaging offre una stretta integrazione con altri AWS servizi. La conoscenza dei seguenti servizi è utile per HealthImaging sfruttarli appieno.

- AWS Identity and Access Management— Utilizza IAM per gestire in modo sicuro le identità e l'accesso alle risorse. HealthImaging
- Amazon Simple Storage Service: usa Amazon S3 come area di staging in cui importare dati DICOM. HealthImaging
- Amazon CloudWatch: CloudWatch da utilizzare per osservare e monitorare HealthImaging le risorse.
- AWS CloudTrail— Utilizzato CloudTrail per tenere traccia HealthImaging dell'attività degli utenti e dell'utilizzo delle API.
- AWS CloudFormation— Utilizzabile AWS CloudFormation per implementare modelli Infrastructure as Code (IaC) in HealthImaging cui creare risorse.
- AWS PrivateLink— Usa Amazon VPC per stabilire la connettività tra Amazon HealthImaging e Amazon Virtual Private Cloud senza esporre i dati a Internet.
- Amazon EventBridge: EventBridge da utilizzare per creare applicazioni scalabili e basate sugli eventi creando regole che indirizzano HealthImaging gli eventi verso le destinazioni.

Accesso ad AWS HealthImaging

Puoi accedere ad AWS HealthImaging utilizzando AWS Management Console gli AWS Command Line Interface e gli AWS SDK. Questa guida fornisce istruzioni procedurali AWS Management Console ed esempi di codice per gli SDK AWS CLI and AWS .

AWS Management Console

AWS Management Console Fornisce un'interfaccia utente basata sul Web per la gestione HealthImaging e le risorse associate. Se hai registrato un AWS account, puoi accedere alla HealthImaging console.

AWS Command Line Interface (AWS CLI)

AWS CLI Fornisce comandi per un'ampia gamma di AWS prodotti ed è supportato su Windows, Mac e Linux. Per ulteriori informazioni, consulta la Guida per l'utente AWS Command Line Interface .

AWS SDK

AWS Gli SDK forniscono librerie, esempi di codice e altre risorse per gli sviluppatori di software. Queste librerie forniscono funzioni di base che automatizzano attività come la firma crittografica

delle richieste, il ritentativo delle richieste e la gestione delle risposte agli errori. Per ulteriori informazioni, consulta Strumenti per creare in AWS.

Richieste HTTP

È possibile HealthImaging eseguire azioni utilizzando richieste HTTP, ma è necessario specificare endpoint diversi a seconda del tipo di azioni utilizzate. Per ulteriori informazioni, consulta Azioni API supportate per le richieste HTTP.

Idoneità alla normativa HIPAA e sicurezza dei dati

Questo è un servizio idoneo ai fini HIPAA. Per ulteriori informazioni sull' AWS U.S. Health Insurance Portability and Accountability Act del 1996 (HIPAA) e sull'utilizzo AWS dei servizi per elaborare, archiviare e trasmettere informazioni sanitarie protette (PHI), vedere Panoramica HIPAA.

Le connessioni HealthImaging contenenti PHI e informazioni di identificazione personale (PII) devono essere crittografate. Per impostazione predefinita, tutte le connessioni HealthImaging utilizzano HTTPS su TLS. HealthImaging archivia i contenuti crittografati dei clienti e opera secondo il modello di responsabilitàAWS condivisa.

Per informazioni sulla conformità, vedereConvalida della conformità per AWS HealthImaging.

Prezzi

HealthImaging ti aiuta ad automatizzare la gestione del ciclo di vita dei dati clinici con la suddivisione in più livelli intelligente. Per ulteriori informazioni, consulta Comprensione dei livelli di storage.

Per informazioni generali sui prezzi, consulta i HealthImaging prezzi di AWS. Per stimare i costi, usa il calcolatore HealthImaging dei prezzi di AWS.

Guida introduttiva ad AWS HealthImaging

Per iniziare a usare AWS HealthImaging, configura un AWS account e crea un AWS Identity and Access Management utente. Per utilizzare gli SDK AWS CLIo AWS gli SDK, devi installarli e configurarli.

Dopo aver appreso HealthImaging i concetti e la configurazione, è disponibile un breve tutorial con esempi di codice per aiutarti a iniziare.

Argomenti

HealthImaging Concetti di AWS

La terminologia e i concetti seguenti sono fondamentali per la comprensione e l'uso di AWS HealthImaging.

Concetti

Datastore

Un data store è un archivio di dati di imaging medico che si trova all'interno di un unico archivio. Regione AWS Un AWS account può avere zero o molti archivi dati. Un data store dispone di una propria chiave di AWS KMS crittografia, quindi i dati di un archivio dati possono essere isolati fisicamente e logicamente dai dati di altri archivi dati. I data store supportano il controllo degli accessi utilizzando ruoli IAM, autorizzazioni e controllo degli accessi basato sugli attributi.

Per ulteriori informazioni, consultare Gestione degli archivi di dati e Comprensione dei livelli di storage.

Set di immagini

Un set di immagini è un AWS concetto che definisce un meccanismo di raggruppamento astratto per l'ottimizzazione dei dati correlati alle immagini mediche. Quando importi i dati di imaging DICOM P10 in un data store AWS, HealthImaging questi vengono trasformati in set di immagini composti da metadati e frame di immagini (dati pixel). L'importazione di dati DICOM P10 produce set di immagini che contengono metadati DICOM e frame di immagini per una o più istanze di Service-Object Pair (SOP) della stessa serie DICOM.

Per ulteriori informazioni, consultare Importazione di dati di imaging e Comprendere i set di immagini.

Metadati

I metadati sono gli attributi non relativi ai pixel presenti all'interno di un set di immagini. Per DICOM, ciò include i dati demografici dei pazienti, i dettagli della procedura e altri parametri specifici dell'acquisizione. AWS HealthImaging separa il set di immagini in metadati e frame di immagini (dati pixel) in modo che le applicazioni possano accedervi rapidamente. Ciò è utile per i visualizzatori di immagini, l'analisi e i casi d'uso di AI/ML che non richiedono dati di pixel. I dati DICOM si normalizzano a livello di paziente, studio e serie, eliminando le incongruenze. Ciò semplifica l'uso dei dati, aumenta la sicurezza e migliora le prestazioni di accesso.

Per ulteriori informazioni, consultare Ottenere i metadati del set di immagini e Normalizzazione dei metadati.

Cornice dell'immagine

Una cornice di immagine è costituita dai dati di pixel presenti all'interno di un set di immagini per creare un'immagine medica 2D. Durante l'importazione, AWS HealthImaging codifica tutti i fotogrammi dell'immagine in formato JPEG 2000 (HTJ2K) ad alta velocità. Pertanto, i fotogrammi delle immagini devono essere decodificati prima della visualizzazione.

Per ulteriori informazioni, consultare Acquisizione dei dati dei pixel del set di immagini e Librerie di decodifica HTJ2K.

Configurazione di AWS HealthImaging

È necessario configurare l' AWS ambiente prima di utilizzare AWS HealthImaging. I seguenti argomenti sono prerequisiti per il tutorial disponibile nella sezione successiva.

Argomenti

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center

2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

Crea bucket S3

Per importare dati DICOM P10 in AWS HealthImaging, sono consigliati due bucket Amazon S3. Il bucket di input Amazon S3 memorizza i dati DICOM P10 da importare e HealthImaging li legge da questo bucket. Il bucket di output di Amazon S3 memorizza i risultati di elaborazione del processo di importazione e HealthImaging scrive in questo bucket. Per una rappresentazione visiva di ciò, consulta il diagramma in. Comprendere i lavori di importazione

> **Note**
>
> A causa della politica AWS Identity and Access Management (IAM), i nomi dei bucket Amazon S3 devono essere univoci. Per ulteriori informazioni, consulta Regole per la denominazione dei bucket nella Guida per l'utente di Amazon Simple Storage Service.

Ai fini di questa guida, specifichiamo i seguenti bucket di input e output di Amazon S3 nel ruolo IAM per l'importazione.

- Bucket di input: `arn:aws:s3:::medical-imaging-dicom-input`
- Secchio di uscita: `arn:aws:s3:::medical-imaging-output`

Per ulteriori informazioni, consulta Creating a bucket nella Amazon S3 User Guide.

Crea un data store

Quando importi i dati di imaging medicale, il HealthImaging data store AWS contiene i risultati dei file DICOM P10 trasformati, chiamati set di immagini. Per una rappresentazione visiva di ciò, consulta il diagramma all'indirizzo. Comprendere i lavori di importazione

> **Tip**
>
> A `datastoreID` viene generato quando si crea un archivio dati. È necessario utilizzare il `datastoreID` quando si completa trust relationship l'importazione più avanti in questa sezione.

Per creare un archivio dati, vedereCreazione di un archivio dati.

Crea un utente IAM con autorizzazione di accesso HealthImaging completa

> **Best practice**
>
> Ti suggeriamo di creare utenti IAM separati per esigenze diverse come l'importazione, l'accesso ai dati e la gestione dei dati. Ciò è in linea con Grant Least Privilege Access nel Well-Architected AWS Framework.
> Ai fini del Tutorial nella prossima sezione, utilizzerai un singolo utente IAM.

Per creare un utente IAM

1. Segui le istruzioni per creare un utente IAM nel tuo AWS account nella Guida per l'utente IAM. Valuta la possibilità di assegnare un nome all'utente `ahiadmin` (o un nome simile) a scopo di chiarimento.
2. Assegna la policy `AWSHealthImagingFullAccess` gestita all'utente IAM. Per ulteriori informazioni, consulta AWSpolitica gestita: AWSHealthImagingFullAccess.

   > **Note**
   >
   > Le autorizzazioni IAM possono essere limitate. Per ulteriori informazioni, consulta AWSpolicy gestite per AWS HealthImaging.

Crea un ruolo IAM per l'importazione

> **Note**
>
> Le seguenti istruzioni si riferiscono a un ruolo AWS Identity and Access Management (IAM) che concede l'accesso in lettura e scrittura ai bucket Amazon S3 per l'importazione dei dati DICOM. Sebbene il ruolo sia richiesto per il tutorial riportato nella sezione successiva, ti consigliamo di aggiungere le autorizzazioni IAM a utenti, gruppi e ruoliAWSpolicy gestite per AWS HealthImaging, poiché sono più facili da usare rispetto alla scrittura delle politiche da soli.

Un ruolo IAM è un'identità IAM che puoi creare nel tuo account e che dispone di autorizzazioni specifiche. Per avviare un processo di importazione, il ruolo IAM che richiama l'`StartDICOMImportJob`azione deve essere associato a una policy utente che conceda l'accesso ai bucket Amazon S3 utilizzati per leggere i dati DICOM P10 e archiviare i risultati dell'elaborazione del processo di importazione. Deve inoltre essere assegnata una relazione di fiducia (policy) che HealthImaging consenta ad AWS di assumere il ruolo.

Per creare un ruolo IAM a fini di importazione

1. Utilizzando la console IAM, crea un ruolo denominato`ImportJobDataAccessRole`. Utilizzerai questo ruolo per il tutorial nella sezione successiva. Per ulteriori informazioni, consulta Creazione di ruoli IAM nella Guida per l'utente IAM .

   > **Tip**
   >
   > Ai fini di questa guida, gli esempi di codice in questione Avvio di un processo di importazione fanno riferimento al ruolo `ImportJobDataAccessRole` IAM.

2. Allega una policy di autorizzazione IAM al ruolo IAM. Questa politica di autorizzazione consente l'accesso ai bucket di input e output di Amazon S3. Allega la seguente politica di autorizzazione al ruolo IAM. `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. Allega la seguente relazione di fiducia (policy) al ruolo `ImportJobDataAccessRole` IAM. La policy di fiducia richiede `datastoreId` che ciò sia stato generato quando hai completato la sezioneCrea un data store. Il tutorial che segue questo argomento presuppone che tu stia utilizzando un HealthImaging data store AWS, ma con bucket Amazon S3 specifici per il data store, ruoli IAM e policy di fiducia.

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

Per ulteriori informazioni sulla creazione e l'utilizzo di policy IAM con AWS HealthImaging, consultaIdentity and Access Management per AWS HealthImaging.

Per saperne di più sui ruoli IAM in generale, consulta i ruoli IAM nella IAM User Guide. Per saperne di più sulle policy e le autorizzazioni IAM in generale, consulta IAM Policies and Permissions nella IAM User Guide.

Installa il AWS CLI (opzionale)

La procedura seguente è necessaria se si utilizza il AWS Command Line Interface. Se utilizzi i AWS Management Console o gli AWS SDK, puoi saltare la procedura seguente.

Per configurare il AWS CLI

1. Scarica e configura la AWS CLI. Per le istruzioni, consulta i seguenti argomenti nella Guida per l'utente di AWS Command Line Interface .

   - Installazione o aggiornamento della versione più recente di AWS CLI
   - Guida introduttiva a AWS CLI

2. Nel AWS CLI `config` file, aggiungi un profilo denominato per l'amministratore. Questo profilo viene utilizzato quando si eseguono i AWS CLI comandi. In base al principio di sicurezza del privilegio minimo, ti consigliamo di creare un ruolo IAM separato con privilegi specifici per le attività eseguite. Per ulteriori informazioni sui profili denominati, consulta Configurazione e impostazioni dei file di credenziali nella Guida per l'AWS Command Line Interface utente.

   ```
   [default]
   ```

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

3. Verificate la configurazione utilizzando il seguente `help` comando.

```
aws medical-imaging help
```

Se AWS CLI è configurato correttamente, viene visualizzata una breve descrizione di AWS HealthImaging e un elenco di comandi disponibili.

HealthImaging Tutorial AWS

Obiettivo

L'obiettivo di questo tutorial è importare file DICOM P10 in un HealthImaging data store AWS e trasformarli in set di immagini composti da metadati e frame di immagini (dati pixel). Dopo aver importato i dati DICOM, accedi ai set di immagini, ai metadati e ai frame di immagini in base alle tue preferenze di accesso a. HealthImaging

Prerequisiti

Tutte le procedure elencate in Configurazione sono necessarie per completare questo tutorial.

Passaggi del tutorial

1. Avvia il processo di importazione
2. Ottieni le proprietà del lavoro di importazione
3. Cerca set di immagini
4. Ottieni le proprietà del set di immagini
5. Ottieni i metadati del set di immagini
6. Ottieni i dati dei pixel del set di immagini
7. Elimina l'archivio dati

Gestione degli archivi di dati con AWS HealthImaging

Con AWS HealthImaging, crei e gestisci archivi di dati per risorse di immagini mediche. I seguenti argomenti descrivono come utilizzare HealthImaging le azioni per creare, descrivere, elencare ed eliminare gli archivi di dati utilizzando AWS Management Console gli e AWS CLI AWS gli SDK.

> **Note**
>
> L'ultimo argomento di questo capitolo riguarda la comprensione dei livelli di storage. Dopo aver importato i dati di imaging medicale in un data store, questi si spostano automaticamente tra due livelli di storage in base al tempo e all'utilizzo. I livelli di storage hanno diversi livelli di prezzo, quindi è importante comprendere il processo di spostamento dei livelli e le HealthImaging risorse riconosciute ai fini della fatturazione.

Argomenti

Creazione di un archivio dati

Usa l'`CreateDatastore`eazione per creare un HealthImaging data store AWS per importare file DICOM P10. I seguenti menu forniscono una procedura e alcuni esempi di codice per gli SDK AWS Management Console and. AWS CLI AWS Per ulteriori informazioni, `CreateDatastore`consulta AWS HealthImaging API Reference.

> **Importante**
>
> Non nominare archivi di dati con informazioni sanitarie protette (PHI), informazioni di identificazione personale (PII) o altre informazioni riservate o sensibili.

Per creare un archivio dati

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina Crea archivio dati della HealthImaging console.
2. In Dettagli, per Nome del Data store, inserisci un nome per il tuo Data Store.
3. In Crittografia dei dati, scegli una AWS KMS chiave per crittografare le tue risorse. Per ulteriori informazioni, consulta Protezione dei dati in AWS HealthImaging.
4. In Tag, facoltativo, puoi aggiungere tag al tuo data store quando lo crei. Per ulteriori informazioni, consulta Taggare una risorsa.
5. Scegli Crea archivio dati.

AWS CLI e SDK

Bash

AWS CLI con script Bash

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

- Per i dettagli sull'API, vedere CreateDatastorein AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per creare un archivio dati

Il seguente esempio di `create-datastore` codice crea un archivio dati con il nome`my-datastore`.

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

Output:

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

Per ulteriori informazioni, consulta Creazione di un data store nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta CreateDatastore AWS CLICommand Reference.

Java

SDK per Java 2.x

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

       return "";
   }
```

- Per i dettagli sull'API, CreateDatastoreconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [CreateDatastore](#)Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Python

SDK per Python (Boto3)

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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta CreateDatastore AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottenere le proprietà del data store

Usa l'`GetDatastore`azione per recuperare le proprietà del HealthImaging data store AWS. I seguenti menu forniscono una procedura AWS Management Console e alcuni esempi di codice per gli SDK AWS CLI and AWS . Per ulteriori informazioni, consulta `GetDatastore`l'AWS HealthImaging API Reference.

Per ottenere le proprietà del data store

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Viene visualizzata la pagina dei dettagli del Data store. Nella sezione Dettagli, sono disponibili tutte le proprietà del data store. Per visualizzare i set di immagini, le importazioni e i tag associati, scegliete la scheda applicabile.

AWS CLI e SDK

Bash

AWS CLI con lo script Bash

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

- Per i dettagli sull'API, vedere GetDatastorein AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per ottenere le proprietà di un archivio dati

Il seguente esempio di `get-datastore` codice ottiene le proprietà di un data store.

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

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

Output:

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

Per ulteriori informazioni, consulta Ottenere le proprietà del data store nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta GetDatastore AWS CLICommand Reference.

Java

SDK per Java 2.x

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

        return null;
    }
```

- Per i dettagli sull'API, GetDatastoreconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

```
  //   }
  // }
  return response["datastoreProperties"];
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetDatastoreReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

```
            raise
        else:
            return data_store["datastoreProperties"]
```

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetDatastore AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elencare archivi di dati

Usa l'`ListDatastores`azione per elencare gli archivi di dati disponibili in AWS HealthImaging. I seguenti menu forniscono una procedura AWS Management Console e alcuni esempi di codice per gli AWS SDK AWS CLI and. Per ulteriori informazioni, `ListDatastores`consulta AWS HealthImaging API Reference.

Per elencare gli archivi di dati

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

- Apri la pagina degli archivi dati della HealthImaging console.

  Tutti gli archivi dati sono elencati nella sezione Archivi dati.

AWS CLI e SDK

Bash

AWS CLI con script Bash

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

- Per i dettagli sull'API, vedere ListDatastoresin AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per elencare gli archivi di dati

Il seguente esempio di `list-datastores` codice elenca gli archivi dati disponibili.

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

Output:

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

Per ulteriori informazioni, consulta Elencare gli archivi di dati nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta ListDatastores AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

           return datastoreSummaries;
       } catch (MedicalImagingException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

- Per i dettagli sull'API, ListDatastoresconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
```

```
  // {
  //   '$metadata': {
  //       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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [ListDatastores](#)Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Python

SDK per Python (Boto3)

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

```
        self.health_imaging_client = health_imaging_client


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListDatastores AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminazione di un archivio dati

Usa l'`DeleteDatastore`azione per eliminare un HealthImaging data store AWS. I seguenti menu forniscono una procedura AWS Management Console e alcuni esempi di codice per gli AWS SDK AWS CLI and. Per ulteriori informazioni, `DeleteDatastore`consulta AWS HealthImaging API Reference.

> **Note**
>
> Prima di poter eliminare un data store, devi prima eliminare tutti i set di immagini al suo interno. Per ulteriori informazioni, consulta Eliminazione di un set di immagini.

Per eliminare un archivio dati

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.
3. Scegli Elimina.

   Viene visualizzata la pagina Elimina data store.
4. Per confermare l'eliminazione del data store, inserisci il nome del data store nel campo di immissione del testo.
5. Scegli Elimina archivio dati.

AWS CLI e SDK

Bash

AWS CLI con lo script Bash

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

- Per i dettagli sull'API, vedere DeleteDatastorein AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per eliminare un archivio dati

Il seguente esempio di `delete-datastore` codice elimina un data store.

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

Output:

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

Per ulteriori informazioni, consulta Eliminazione di un data store nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta DeleteDatastore AWS CLICommand Reference.

Java

SDK per Java 2.x

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

- Per i dettagli sull'API, DeleteDatastoreconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteDatastoreReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta DeleteDatastore AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Comprensione dei livelli di storage

AWS HealthImaging utilizza la suddivisione in più livelli intelligente per la gestione automatica del ciclo di vita clinico. Ciò si traduce in prestazioni e prezzi convincenti sia per i dati nuovi o attivi che per i dati di archiviazione a lungo termine senza problemi. HealthImaging fatturazione dello storage per GB/mese utilizzando i seguenti livelli.

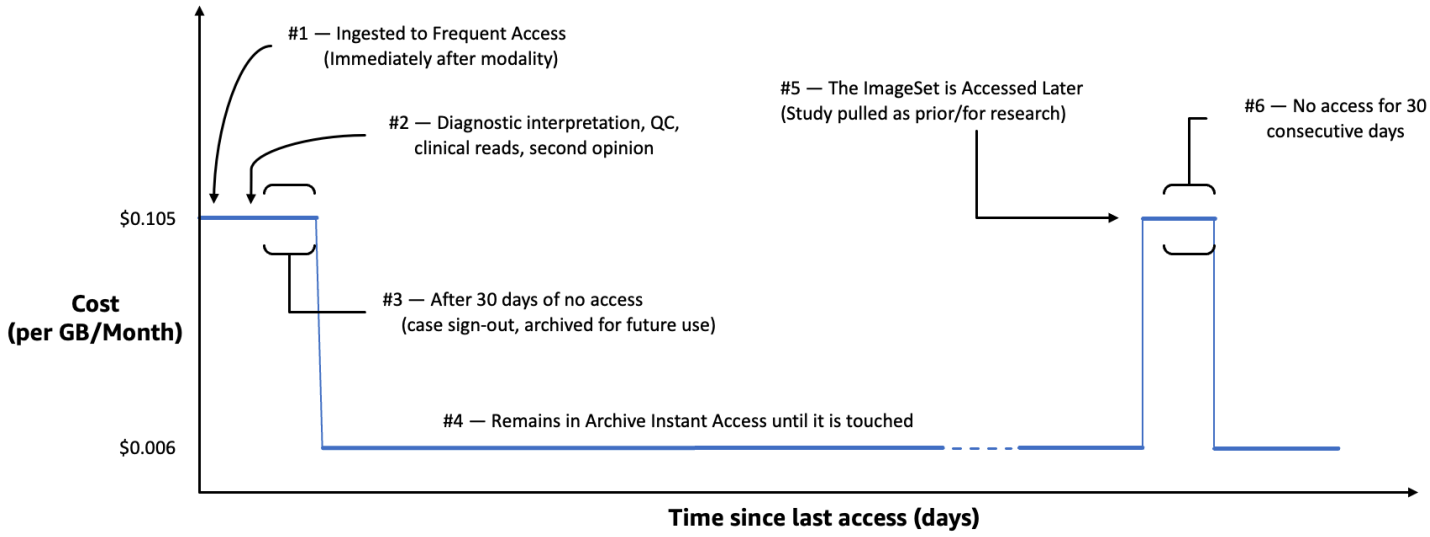
- Livello di accesso frequente: un livello per i dati a cui si accede di frequente.
- Archive Instant Access Tier: un livello per i dati archiviati.

> **Note**
>
> Non vi è alcuna differenza di prestazioni tra i livelli Frequent Access e Archive Instant Access. Il tiering intelligente viene applicato a azioni API specifiche del set di immagini. La tiering intelligente non riconosce le azioni API di archiviazione, importazione e etichettatura dei dati. Lo spostamento tra i livelli è automatico in base all'utilizzo dell'API ed è spiegato nella sezione seguente.

Come funziona lo spostamento dei livelli?

- Dopo l'importazione, i set di immagini iniziano nel livello di accesso frequente.
- Dopo 30 giorni consecutivi senza modifiche, i set di immagini passano automaticamente al livello Archive Instant Access.
- I set di immagini in Archive Instant Access Tier tornano al livello di accesso frequente solo dopo essere stati toccati.

Il grafico seguente fornisce una panoramica del processo di tiering HealthImaging intelligente.



Cosa è considerato un tocco?

Un tocco è un accesso specifico all'API tramite AWS Management Console AWS CLI, o AWS SDK e si verifica quando:

1. Viene creato un nuovo set di immagini (`StartDICOMImportJoboCopyImageSet`)
2. Un set di immagini viene aggiornato (`UpdateImageSetMetadataoCopyImageSet`)
3. I metadati o il frame di immagine associati a un set di immagini (dati pixel) vengono letti (`GetImageSetMetaDataoGetImageFrame`)

Le seguenti azioni HealthImaging API comportano tocchi e spostano i set di immagini dal livello Archive Instant Access al Frequent Access Tier.

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

> **Note**
>
> Sebbene i frame delle immagini (dati sui pixel) non possano essere eliminati utilizzando l'`UpdateImageSetMetadata`azione, vengono comunque conteggiati ai fini della fatturazione.

Le seguenti azioni HealthImaging API non comportano tocchi. Pertanto, non spostano i set di immagini dal livello Archive Instant Access al Frequent Access Tier.

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

Importazione di dati di imaging con AWS HealthImaging

L'importazione è il processo di trasferimento dei dati di imaging medicale da un bucket di input Amazon S3 a un data store AWS HealthImaging. Durante l'importazione, AWS HealthImaging esegue un controllo di verifica dei dati dei pixel prima di trasformare i file DICOM P10 in set di immagini composti da metadati e frame di immagini (dati pixel).

> **Tip**
>
> Dopo aver acquisito dimestichezza con l'implementazione HealthImaging, ti invitiamo a visitare il sito per iniziare subito Progetti HealthImaging di esempio AWS a utilizzare i nostri progetti di importazione e visualizzazione.

I seguenti argomenti descrivono come importare i dati di imaging medicale in un HealthImaging data store utilizzandoAWS Management Console, AWS CLI e gli SDK. AWS

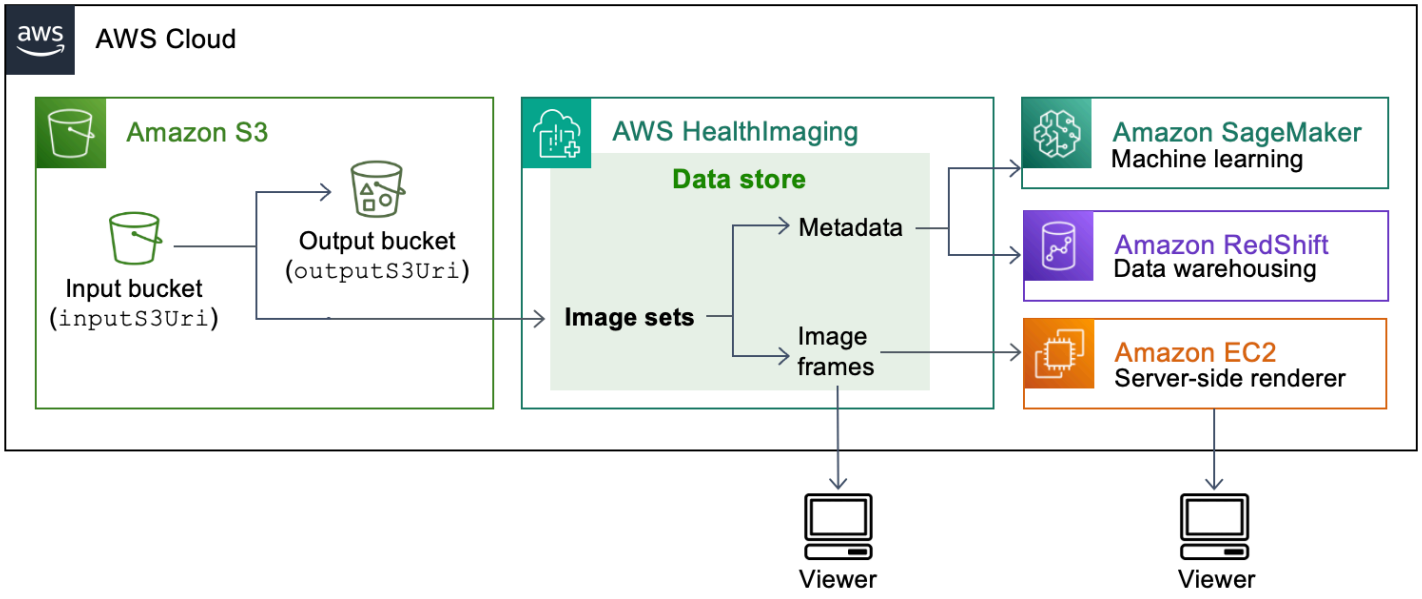
Argomenti

- Comprendere i lavori di importazione
- Avvio di un processo di importazione
- Ottenere proprietà lavorative da importare
- Elencare lavori di importazione

Comprendere i lavori di importazione

Dopo aver creato un data store in AWS HealthImaging, devi importare i dati di imaging medicale dal bucket di input Amazon S3 nel tuo data store per creare set di immagini. Puoi utilizzare gli AWS Management Console AWS CLI, e gli AWS SDK per avviare, descrivere ed elencare i lavori di importazione.

Il diagramma seguente fornisce una panoramica di come HealthImaging importare i dati DICOM in un data store e trasformarli in set di immagini. I risultati dell'elaborazione dei processi di importazione vengono archiviati nel bucket di output di Amazon S3 (`outputS3Uri`) e i set di immagini vengono archiviati nel data store HealthImaging AWS.



Tieni a mente i seguenti punti quando importi i tuoi file di immagini mediche da Amazon S3 in un data store HealthImaging AWS:

- Per i lavori di importazione sono supportate classi SOP e sintassi di trasferimento specifiche. Per ulteriori informazioni, consulta Supporto DICOM.
- I vincoli di lunghezza si applicano a elementi DICOM specifici durante l'importazione. Per garantire una corretta operazione di importazione, verificate che i dati di imaging medicale non superino i limiti di lunghezza. Per ulteriori informazioni, consulta vincoli degli elementi DICOM.
- All'inizio dei processi di importazione viene eseguito un controllo di verifica dei dati relativi ai pixel. Per ulteriori informazioni, consulta Verifica dei dati relativi ai pixel.
- Esistono endpoint, quote e limiti di limitazione associati alle azioni di importazione. HealthImaging Per ulteriori informazioni, consulta Endpoint e quote e Limiti di limitazione.
- Per ogni processo di importazione, i risultati di elaborazione vengono archiviati nella posizione. `outputS3Uri` I risultati dell'elaborazione sono organizzati in `job-output-manifest.json` file `SUCCESS` e `FAILURE` cartelle.

> **Note**
>
> È possibile includere fino a 10.000 cartelle annidate per un singolo processo di importazione.

- Il `job-output-manifest.json` file contiene `jobSummary` output e dettagli aggiuntivi sui dati elaborati. L'esempio seguente mostra l'output di un `job-output-manifest.json` file.

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

- La SUCCESS cartella contiene il `success.ndjson` file contenente i risultati di tutti i file di immagine importati correttamente. L'esempio seguente mostra l'output di un `success.ndjson` file.

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

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

- La `FAILURE` cartella contiene il `failure.ndjson` file contenente i risultati di tutti i file di immagine che non sono stati importati correttamente. L'esempio seguente mostra l'output di un `failure.ndjson` file.

```
{"inputFile":"dicom_input/invalidDicomFile1.dcm","exception":
{"exceptionType":"ValidationException","message":"DICOM attribute TransferSyntaxUID
 does not exist"}}
{"inputFile":"dicom_input/invalidDicomFile2.dcm","exception":
{"exceptionType":"ValidationException","message":"DICOM attributes does not
 exist"}}
```

- I lavori di importazione vengono conservati nell'elenco dei lavori per 90 giorni e quindi archiviati.

Avvio di un processo di importazione

Usa l'`StartDICOMImportJob`azione per avviare un controllo di verifica dei dati dei pixel e importare dati in blocco in un HealthImaging data store AWS. Il processo di importazione importa i file DICOM P10 che si trovano nel bucket di input di Amazon S3 specificato dal parametro. `inputS3Uri` I risultati dell'elaborazione del processo di importazione vengono archiviati nel bucket di output di Amazon S3 specificato dal parametro. `outputS3Uri`

> **Note**
>
> HealthImaging supporta l'importazione di dati da bucket Amazon S3 situati in altre regioni supportate. Per ottenere questa funzionalità, fornisci il `inputOwnerAccountId` parametro all'avvio di un processo di importazione. Per ulteriori informazioni, consulta Importazione tra più account per AWS HealthImaging.
> Durante l'importazione, i vincoli di lunghezza vengono applicati a elementi DICOM specifici. Per ulteriori informazioni, consulta vincoli degli elementi DICOM.

I seguenti menu forniscono una procedura e alcuni esempi di codice per gli SDK AWS Management Console and. AWS CLI AWS Per ulteriori informazioni, consulta `StartDICOMImportJob`l'AWS HealthImaging API Reference.

Per avviare un processo di importazione

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.
3. Scegli Importa dati DICOM.

   Viene visualizzata la pagina Importa dati DICOM.
4. Nella sezione Dettagli, inserisci le seguenti informazioni:

   - Nome (opzionale)
   - Importa la posizione di origine in S3
   - ID dell'account del proprietario del bucket di origine (opzionale)
   - Chiave di crittografia (opzionale)
   - Destinazione di uscita in S3
5. Nella sezione Accesso al servizio, scegli Usa un ruolo di servizio esistente e seleziona il ruolo dal menu Service role name oppure scegli Crea e usa un nuovo ruolo di servizio.
6. Seleziona Importa.

AWS CLI e SDK

C++

SDK per C++

```
//! Routine which starts a HealthImaging import job.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param inputBucketName: The name of the Amazon S3 bucket containing the DICOM
 files.
  \param inputDirectory: The directory in the S3 bucket containing the DICOM
 files.
```

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

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

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

    return startDICOMImportJobOutcome.IsSuccess();
}
```

- Per i dettagli sull'API, consulta StartDicom ImportJob in AWS SDK for C++ API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per avviare un processo di importazione di file dicom

Il seguente esempio di `start-dicom-import-job` codice avvia un processo di importazione dicom.

```
aws medical-imaging start-dicom-import-job \
    --job-name "my-job" \
    --datastore-id "12345678901234567890123456789012" \
    --input-s3-uri "s3://medical-imaging-dicom-input/dicom_input/" \
    --output-s3-uri "s3://medical-imaging-output/job_output/" \
    --data-access-role-arn "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole"
```

Output:

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

Per ulteriori informazioni, consulta Avvio di un processo di importazione nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta StartDicom ImportJob in AWS CLI Command Reference.

Java

SDK per Java 2.x

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

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

       return "";
   }
```

- Per i dettagli sull'API, consulta StartDicom ImportJob in API Reference.AWS SDK for Java 2.x

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

- Per i dettagli sull'API, consulta StartDicom ImportJob in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta StartDICOM ImportJob nella guida di riferimento all'API AWS SDK for Python (Boto3).

> **Note**
>
> C' GitHubè di più su. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottenere proprietà lavorative da importare

Usa l'`GetDICOMImportJob`azione per saperne di più sulle proprietà dei job di HealthImaging importazione di AWS. Ad esempio, dopo aver avviato un processo di importazione, puoi `GetDICOMImportJob` correre per trovare lo stato del lavoro. Una volta che il file `jobStatus` torna come`COMPLETED`, sei pronto per accedere ai tuoi set di immagini.

> **Note**
>
> `jobStatus`Si riferisce all'esecuzione del processo di importazione. Pertanto, un processo di importazione può restituire un annuncio `jobStatus` `COMPLETED` anche se durante il processo di importazione vengono rilevati problemi di convalida. Se un `jobStatus` restituisce come`COMPLETED`, ti consigliamo comunque di esaminare i manifesti di output scritti in Amazon S3, in quanto forniscono dettagli sull'esito positivo o negativo delle importazioni di singoli oggetti P10.

I seguenti menu forniscono una procedura e alcuni esempi di codice per gli SDK AWS Management Console and. AWS CLI AWS Per ulteriori informazioni, consulta `GetDICOMImportJob`l'AWS HealthImaging API Reference.

Per ottenere le proprietà dei lavori di importazione

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Viene visualizzata la pagina dei dettagli del Data store. La scheda Set di immagini è selezionata per impostazione predefinita.
3. Scegliete la scheda Importazioni.
4. Scegli un lavoro di importazione.

   Viene visualizzata la pagina dei dettagli del processo di importazione con le proprietà relative al processo di importazione.

AWS CLI e SDK

C++

SDK per C++

```
//! Routine which gets a HealthImaging DICOM import job's properties.
```

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

    return outcome;
}
```

- Per i dettagli sull'API, consulta getDICOM ImportJob in AWS SDK for C++ API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per ottenere le proprietà di un processo di importazione dicom

Il seguente esempio di `get-dicom-import-job` codice ottiene le proprietà di un processo di importazione dicom.

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

Output:

```
{
    "jobProperties": {
        "jobId": "09876543210987654321098765432109",
        "jobName": "my-job",
        "jobStatus": "COMPLETED",
        "datastoreId": "12345678901234567890123456789012",
        "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
        "endedAt": "2022-08-12T11:29:42.285000+00:00",
        "submittedAt": "2022-08-12T11:28:11.152000+00:00",
        "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",
        "outputS3Uri": "s3://medical-imaging-output/
job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/"
    }
}
```

Per ulteriori informazioni, consulta Ottenere le proprietà del processo di importazione nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta GetDicom ImportJob in AWS CLI Command Reference.

Java

SDK per Java 2.x

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

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

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

        return null;
    }
```

- Per i dettagli sull'API, consulta getDicom ImportJob in API Reference.AWS SDK for Java 2.x

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

- Per i dettagli sull'API, consulta getDICOM ImportJob in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetDICOM ImportJob nella guida di riferimento all'API AWS SDK for Python (Boto3).

> **Note**
>
> C' GitHubè di più su. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elencare lavori di importazione

Usa l'`ListDICOMImportJobs`azione per elencare i lavori di importazione creati per un HealthImaging data store specifico. I menu seguenti forniscono una procedura AWS Management Console e alcuni esempi di codice per gli AWS SDK AWS CLI and. Per ulteriori informazioni, consulta `ListDICOMImportJobs`l'AWS HealthImaging API Reference.

> **Note**
>
> I lavori di importazione vengono conservati nell'elenco dei lavori per 90 giorni e quindi archiviati.

Per elencare i lavori di importazione

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Viene visualizzata la pagina dei dettagli del Data store. La scheda Set di immagini è selezionata per impostazione predefinita.
3. Scegliete la scheda Importazioni per elencare tutti i lavori di importazione associati.

AWS CLI e SDK

CLI

AWS CLI

Per elencare i lavori di importazione in formato dicom

Il seguente esempio di `list-dicom-import-jobs` codice elenca i processi di importazione dicom.

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

Output:

```
{
    "jobSummaries": [
        {
            "jobId": "09876543210987654321098765432109",
            "jobName": "my-job",
            "jobStatus": "COMPLETED",
            "datastoreId": "12345678901234567890123456789012",
            "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
            "endedAt": "2022-08-12T11:21:56.504000+00:00",
            "submittedAt": "2022-08-12T11:20:21.734000+00:00"
        }
    ]
}
```

Per ulteriori informazioni, consulta Elencare i lavori di importazione nella AWS HealthImaging Developer Guide.

- Per i dettagli sull'API, consulta ListDicom ImportJobs in AWS CLI Command Reference.

Java

SDK per Java 2.x

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

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

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

- Per i dettagli sull'API, consulta ListDicom ImportJobs in API Reference.AWS SDK for Java 2.x

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta ListDicom ImportJobs in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListDicom ImportJobs in AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> C' GitHubè di più su. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Accesso ai set di immagini con AWS HealthImaging

L'accesso ai dati di imaging medico in AWS in HealthImaging genere implica la ricerca di un set di immagini con una chiave univoca e l'ottenimento dei metadati e dei frame di immagine associati (dati pixel).

> **Tip**
>
> Dopo aver acquisito dimestichezza con AWS HealthImaging, ti invitiamo a visitarlo per scoprire come Progetti HealthImaging di esempio AWS iniziare l'implementazione utilizzando i nostri progetti di visualizzazione.

I seguenti argomenti spiegano cosa sono i set di immagini e come utilizzare gli AWS SDK per cercarli e ottenere le proprietà AWS Management Console AWS CLI, i metadati e i frame di immagini associati.

Argomenti

Comprendere i set di immagini

I set di immagini sono un AWS concetto che funge da base per AWS HealthImaging. I set di immagini vengono creati quando si importano i dati DICOM in un computer HealthImaging, quindi è necessario conoscerli bene quando si lavora con il servizio.

I set di immagini sono stati introdotti per i seguenti motivi:

- Supporta un'ampia varietà di flussi di lavoro di imaging medicale (clinici e non clinici) tramite API flessibili.

- Massimizza la sicurezza dei pazienti raggruppando solo i dati correlati.
- Incoraggia la pulizia dei dati per aumentare la visibilità delle incongruenze. Per ulteriori informazioni, consulta Modifica dei set di immagini.

> **Importante**
>
> L'uso clinico dei dati DICOM prima della loro pulizia può causare danni ai pazienti.

I seguenti menu descrivono i set di immagini in modo più dettagliato e forniscono esempi e diagrammi per aiutarvi a comprenderne la funzionalità e lo scopo. HealthImaging

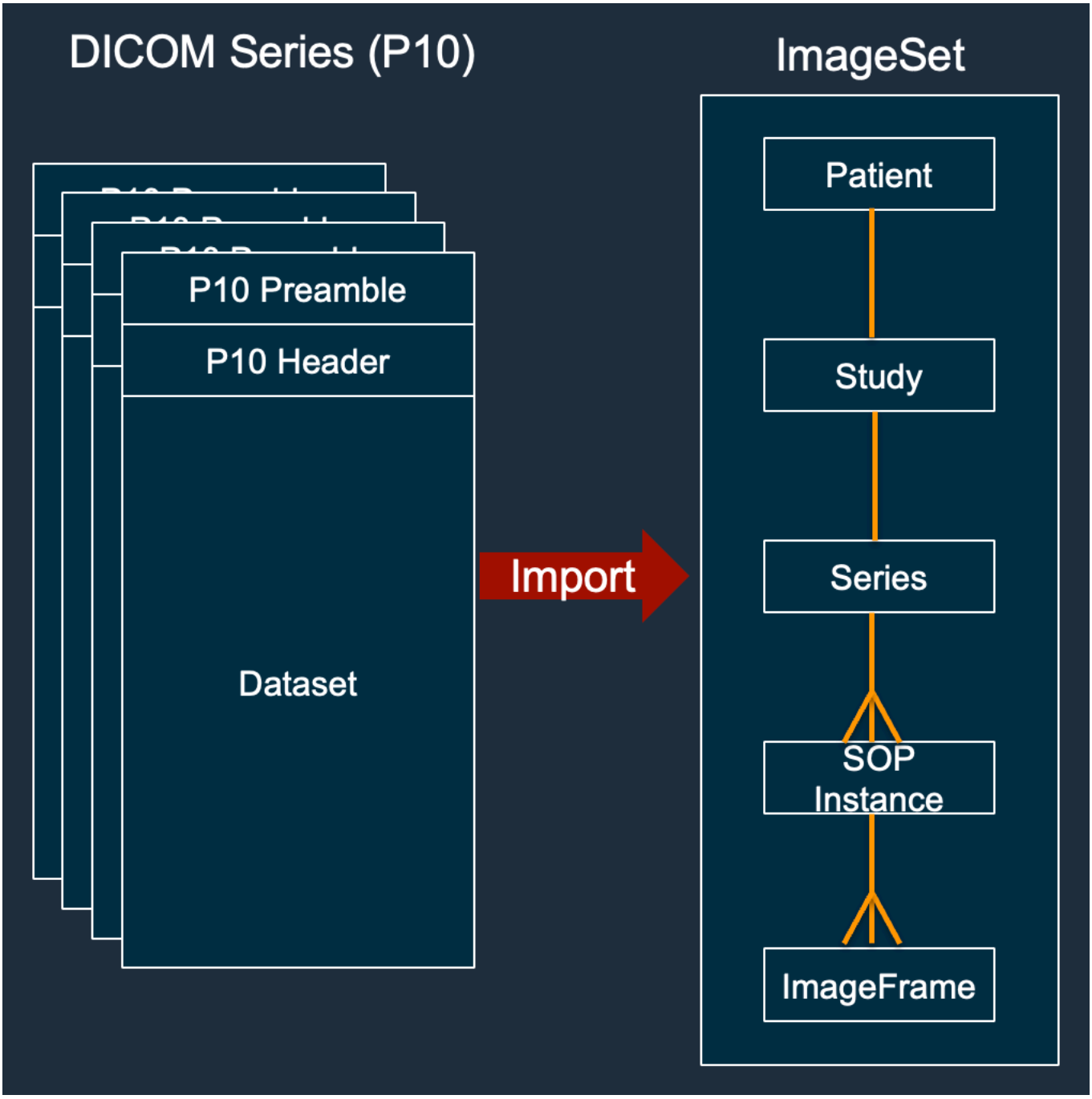
Cos'è un set di immagini?

Un set di immagini è un AWS concetto che definisce un meccanismo di raggruppamento astratto per l'ottimizzazione dei dati correlati alle immagini mediche. Quando importi i dati di imaging DICOM P10 in un data store AWS, HealthImaging questi vengono trasformati in set di immagini composti da metadati e frame di immagini (dati pixel).

> **Note**
>
> I metadati del set di immagini sono normalizzati. In altre parole, un insieme comune di attributi e valori corrisponde agli elementi a livello di paziente, studio e serie elencati nel Registro degli elementi di dati DICOM.
> I fotogrammi delle immagini (dati in pixel) sono codificati in formato JPEG 2000 (HTJ2K) ad alta velocità e devono essere decodificati prima della visualizzazione.

I set di immagini sono AWS risorse, quindi vengono assegnati Amazon Resource Names (ARN). Possono essere etichettati con un massimo di 50 coppie chiave-valore e possono essere garantiti il controllo degli accessi basato sui ruoli (RBAC) e il controllo degli accessi basato sugli attributi (ABAC) tramite IAM. Inoltre, i set di immagini dispongono di versioni, in modo che tutte le modifiche vengano mantenute e sia possibile accedere alle versioni precedenti.

L'importazione di dati DICOM P10 produce set di immagini che contengono metadati DICOM e frame di immagini per una o più istanze di Service-Object Pair (SOP) della stessa serie DICOM.



Note

Processi di importazione DICOM:

- Crea sempre nuovi set di immagini e non aggiornare mai i set di immagini esistenti.
- Non deduplicate lo storage dell'istanza SOP, poiché ogni importazione della stessa istanza SOP utilizza spazio di archiviazione aggiuntivo.

- Può creare più set di immagini per una singola serie DICOM. Ad esempio, quando esiste una variante di un attributo di metadati normalizzato, ad esempio una mancata corrispondenza. `PatientName`

Che aspetto hanno i metadati dei set di immagini?

Utilizzate l'`GetImageSetMetadata`azione per recuperare i metadati del set di immagini. I metadati restituiti vengono compressi con`gzip`, quindi è necessario decomprimerli prima di visualizzarli. Per ulteriori informazioni, consulta Ottenere i metadati del set di immagini.

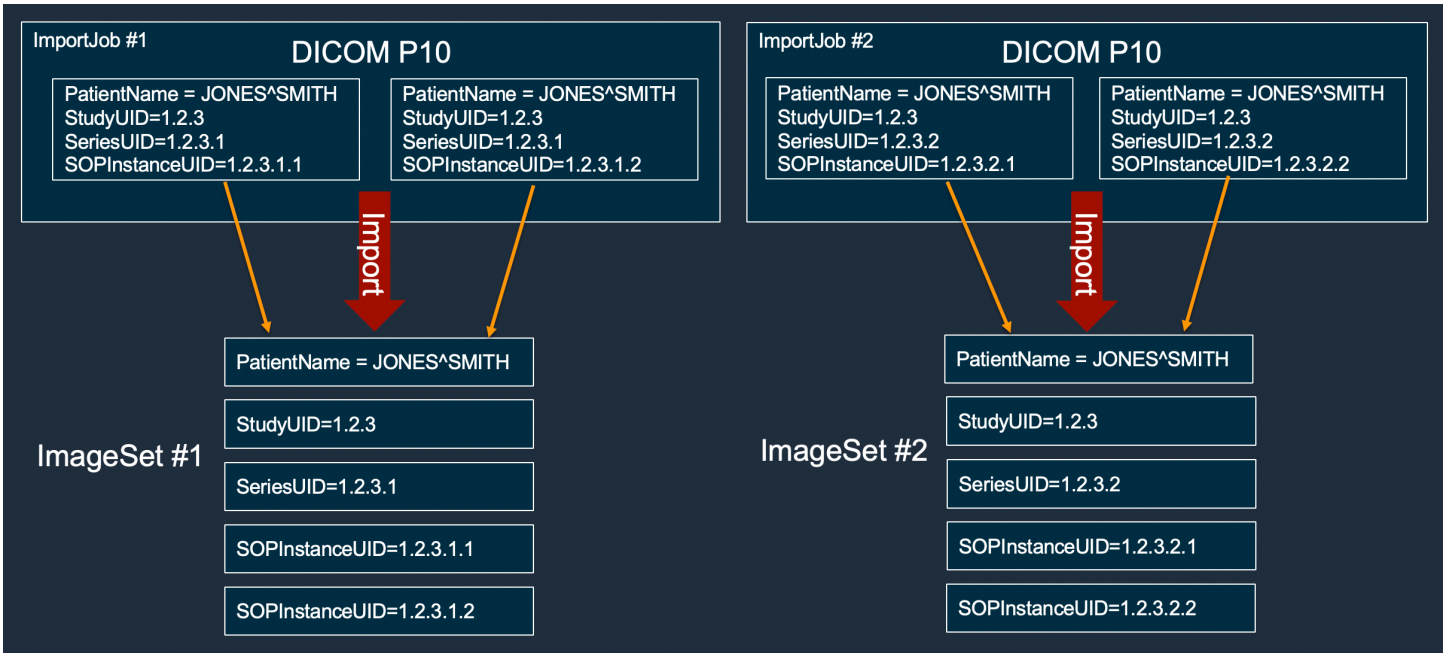
L'esempio seguente mostra la struttura dei metadati del set di immagini in formato JSON.

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

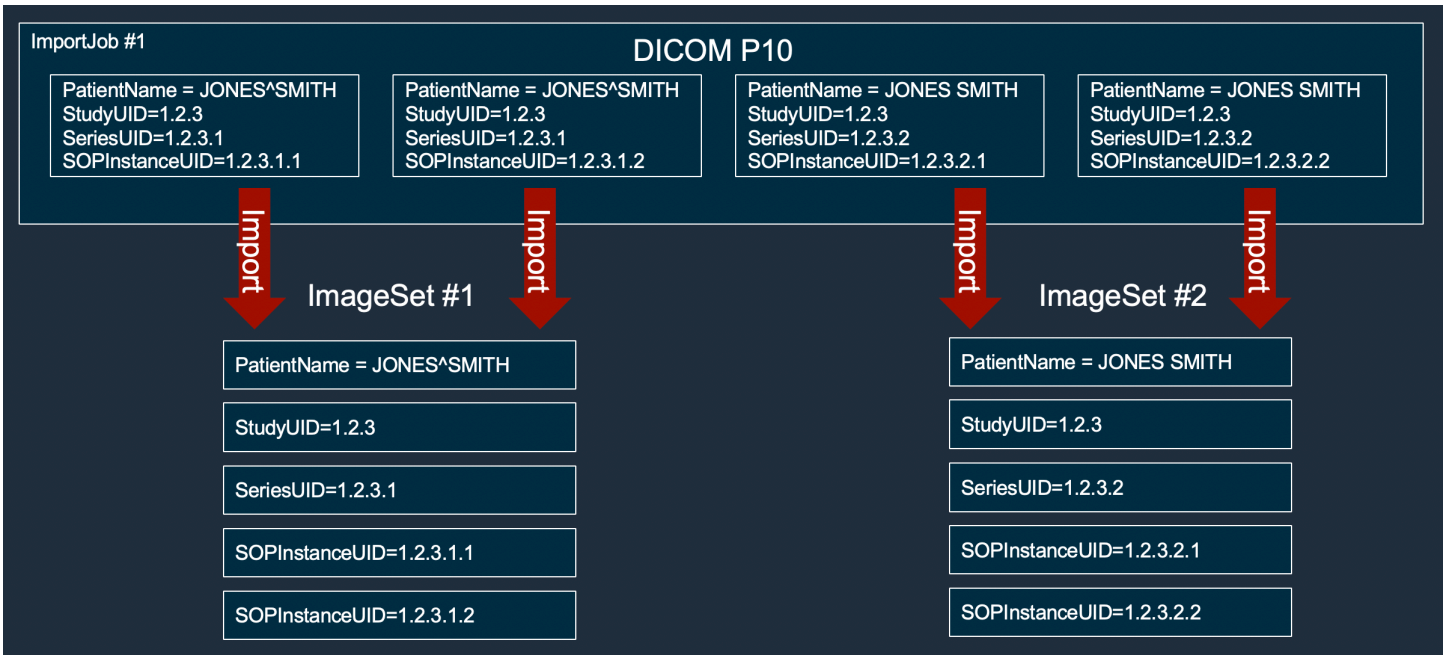
Esempio di creazione di set di immagini: processi di importazione multipli

L'esempio seguente mostra come più processi di importazione creino sempre nuovi set di immagini e non si aggiungano mai a quelli esistenti.



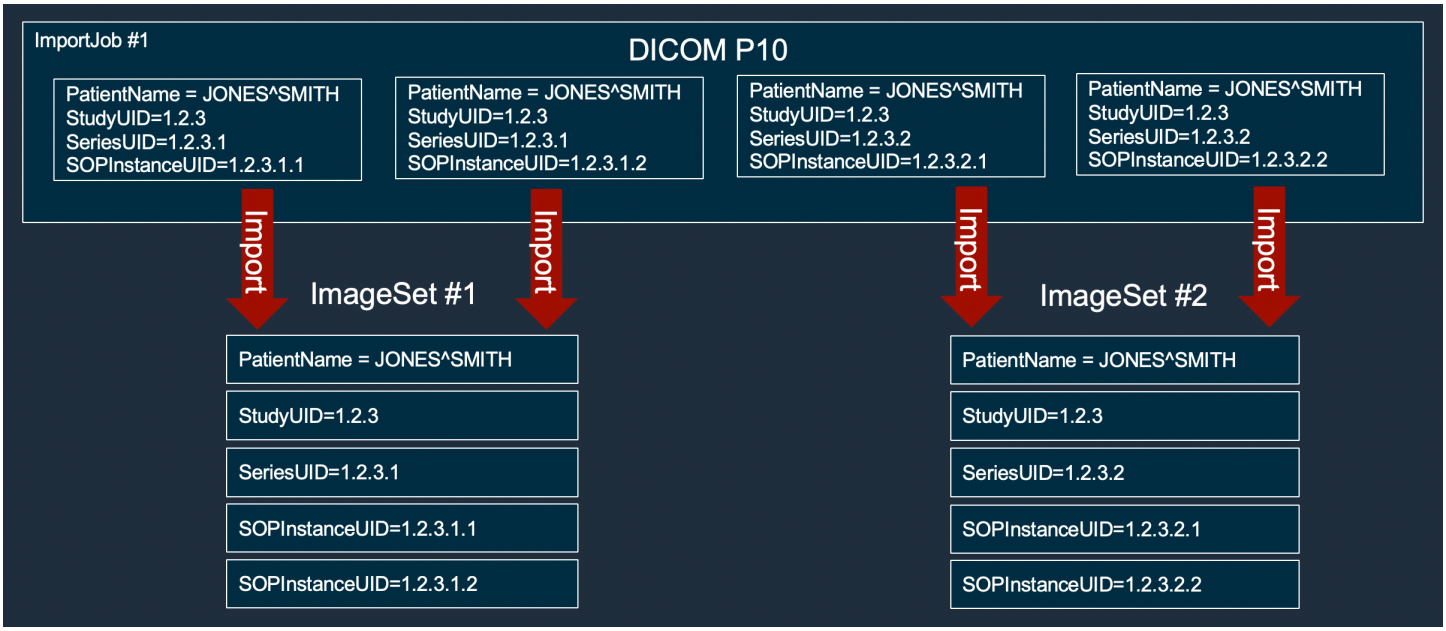
Esempio di creazione di un set di immagini: processo di importazione singolo con due varianti

L'esempio seguente mostra un singolo processo di importazione che crea due set di immagini perché le istanze 1 e 2 hanno nomi di pazienti diversi rispetto alle istanze 3 e 4.



Esempio di creazione di un set di immagini: processo di importazione singolo con ottimizzazione

L'esempio seguente mostra un singolo processo di importazione che crea due set di immagini per migliorare la produttività, anche se i nomi dei pazienti corrispondono.



Ricerca di set di immagini

Usa l'`SearchImageSets`azione per eseguire query di ricerca su tutti i set di immagini in un `ACTIVE` HealthImaging data store. I menu seguenti forniscono una procedura AWS Management Console e alcuni esempi di codice per gli SDK AWS CLI e AWS . Per ulteriori informazioni, `SearchImageSets`consulta AWS HealthImaging API Reference.

> **Note**
>
> Tieni a mente i seguenti punti durante la ricerca di set di immagini.
>
> - `SearchImageSets`accetta un singolo parametro di query di ricerca e restituisce una risposta impaginata di tutti i set di immagini che hanno i criteri corrispondenti. Tutte le interrogazioni relative all'intervallo di date devono essere inserite come. `(lowerBound, upperBound)`
> - Per impostazione predefinita, `SearchImageSets` utilizza il `updatedAt` campo per l'ordinamento decrescente dal più recente al meno recente.

- Se hai creato il tuo data store con una chiave di proprietà del cliente, devi aggiornare la policy AWS KMS AWS KMS chiave prima di interagire con i set di immagini. Per ulteriori informazioni, consulta Creazione di una chiave gestita dal cliente.

Per cercare set di immagini

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

> **Note**
>
> Le seguenti procedure mostrano come cercare set di immagini utilizzando i filtri `Series Instance UID` e le `Updated at` proprietà.

Series Instance UID

Cerca set di immagini utilizzando il filtro delle **`Series Instance UID`** proprietà

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Si apre la pagina dei dettagli del Data Store e la scheda Set di immagini è selezionata per impostazione predefinita.
3. Scegli il menu del filtro delle proprietà e seleziona`Series Instance UID`.
4. Nel campo Inserisci il valore da cercare, inserisci (incolla) l'UID dell'istanza della serie che ti interessa.

   > **Note**
   >
   > I valori Series Instance UID devono essere identici a quelli elencati nel Registry of DICOM Unique Identifiers (UID). Tieni presente che i requisiti includono una serie di numeri che contengano almeno un punto tra di loro. I periodi non sono consentiti all'inizio o alla fine degli UID delle istanze Series. Le lettere e gli spazi bianchi non sono consentiti, quindi fai attenzione quando copi e incolla gli UID.

5. Scegli il menu Intervallo di date, seleziona un intervallo di date per il Series Instance UID e scegli Applica.
6. Selezionare Search (Cerca).

   Gli UID delle istanze Series che rientrano nell'intervallo di date selezionato vengono restituiti nell'ordine più recente per impostazione predefinita.

Updated at

Cerca set di immagini utilizzando il filtro delle proprietà **`Updated at`**

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Si apre la pagina dei dettagli del Data Store e la scheda Set di immagini è selezionata per impostazione predefinita.
3. Scegliete il menu del filtro delle proprietà e sceglieteUpdated at.
4. Scegliete il menu Intervallo di date, selezionate un intervallo di date del set di immagini e scegliete Applica.
5. Selezionare Search (Cerca).

   Per impostazione predefinita, i set di immagini che rientrano nell'intervallo di date selezionato vengono restituiti nell'ordine più recente.

AWS CLI e SDK

C++

SDK per C++

La funzione di utilità per la ricerca di set di immagini.

```
//! 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;
}
```

Caso d'uso #1: operatore EQUAL.

```
       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;
           }
       }
```

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

```
        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;
           }
       }
```

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano stati precedentemente proseguiti.

```
       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;
           }
       }
```

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
       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;
            }
        }
```

- Per i dettagli sull'API, vedere in API Reference. SearchImageSetsAWS SDK for C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Esempio 1: per cercare set di immagini con un operatore EQUAL

Il seguente esempio di `search-image-sets` codice utilizza l'operatore EQUAL per cercare set di immagini in base a un valore specifico.

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

Contenuto di `search-criteria.json`

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

Output:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
             "DICOMPatientSex": "F",
             "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
             "DICOMPatientBirthDate": "19201120",
             "DICOMStudyDescription": "UNKNOWN",
             "DICOMPatientId": "SUBJECT08701",
             "DICOMPatientName": "Melissa844 Huel628",
             "DICOMNumberOfStudyRelatedInstances": 1,
             "DICOMStudyTime": "140728",
             "DICOMNumberOfStudyRelatedSeries": 1
            },
```

```
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

Esempio 2: per cercare set di immagini con un operatore BETWEEN utilizzando DICOM StudyDate e DICOM StudyTime

Il seguente esempio di `search-image-sets` codice cerca set di immagini con DICOM Studies generati tra il 1° gennaio 1990 (00:00) e il 1° gennaio 2023 (00:00).

Nota: DICOM è facoltativo. StudyTime Se non è presente, 12:00 AM (inizio della giornata) è il valore temporale per le date fornite per il filtraggio.

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

Contenuto di `search-criteria.json`

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

Output:

```
{
    "imageSetsMetadataSummaries": [{
```

```
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

Esempio 3: per cercare set di immagini con un operatore BETWEEN utilizzando CreateDat (gli studi temporali erano precedentemente persistenti)

Il seguente esempio di `search-image-sets` codice cerca set di immagini con DICOM Studies persistenti HealthImaging tra gli intervalli di tempo del fuso orario UTC.

Nota: fornire CreatedAt in un formato di esempio («1985-04-12T 23:20:50.52 Z»).

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

Contenuto di `search-criteria.json`

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

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

Output:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

Esempio 4: cercare set di immagini con un operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordinare la risposta in ordine ASC nel campo updatedAt

Il seguente esempio di `search-image-sets` codice cerca i set di immagini con un operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC sul campo updatedAt.

Nota: fornire UpdatedAt in un formato di esempio («1985-04-12T 23:20:50.52 Z»).

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

Contenuto di `search-criteria.json`

```
{
    "filters": [{
        "values": [{
            "updatedAt": "2024-03-11T15:00:05.074000-07:00"
        }, {
            "updatedAt": "2024-03-11T16:00:05.074000-07:00"
        }],
        "operator": "BETWEEN"
    }, {
        "values": [{
            "DICOMSeriesInstanceUID": "1.2.840.99999999.84710745.943275268089"
        }],
        "operator": "EQUAL"
    }],
    "sort": {
        "sortField": "updatedAt",
        "sortOrder": "ASC"
    }
}
```

Output:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
```

```
}
```

Per ulteriori informazioni, consulta Searching image sets nella Developer Guide.AWS HealthImaging

- Per i dettagli sull'API, consulta SearchImageSets AWS CLICommand Reference.

Java

SDK per Java 2.x

La funzione di utilità per la ricerca di set di immagini.

```
   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;
   }
```

Caso d'uso #1: operatore EQUAL.

```
        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();
        }
```

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

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

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

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

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

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

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

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano stati precedentemente proseguiti.

```
        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();
        }
```

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
        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();
        }
```

- Per i dettagli sull'API, vedere in API Reference. SearchImageSetsAWS SDK for Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

La funzione di utilità per la ricerca di set di immagini.

```
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;
};
```

Caso d'uso #1: operatore EQUAL.

```
    const datastoreId = "12345678901234567890123456789012";

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

```
        };

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

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

```
    const datastoreId = "12345678901234567890123456789012";

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

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

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano stati precedentemente proseguiti.

```
    const datastoreId = "12345678901234567890123456789012";

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

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

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
                        {updatedAt: new Date()},
                    ],
                    operator: "BETWEEN",
                },
                {
                    values: [
                        {DICOMSeriesInstanceUID:
 "1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
                    ],
                    operator: "EQUAL",
                },
```

```
            ],
            sort: {
                sortOrder: "ASC",
                sortField: "updatedAt",
            }
        };

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

- Per i dettagli sull'API, vedere in API Reference. SearchImageSetsAWS SDK for JavaScript

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

La funzione di utilità per la ricerca di set di immagini.

```
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
```

Caso d'uso #1: operatore EQUAL.

```
        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}")
```

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

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

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

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

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano stati precedentemente proseguiti.

```
        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}"
        )
```

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
        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}")
```

Il codice seguente crea un'istanza dell'oggetto. MedicalImagingWrapper

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

- Per i dettagli sull'API, consulta SearchImageSets AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottenere le proprietà del set di immagini

Usa l'`GetImageSet`azione per restituire le proprietà per un determinato set di immagini HealthImaging. I menu seguenti forniscono una procedura per gli AWS Management Console ed esempi di codice per gli AWS SDK AWS CLI and. Per ulteriori informazioni, consulta `GetImageSet`l'AWS HealthImaging API Reference.

> **Note**
>
> Per impostazione predefinita, AWS HealthImaging restituisce le proprietà per l'ultima versione di un set di immagini. Per visualizzare le proprietà di una versione precedente di un set di immagini, fornisci `versionId` la tua richiesta.

Per ottenere le proprietà del set di immagini

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

Si apre la pagina dei dettagli del Data Store e la scheda Set di immagini è selezionata per impostazione predefinita.

3. Scegliete un set di immagini.

   Viene visualizzata la pagina dei dettagli del set di immagini con le proprietà del set di immagini.

AWS CLI e SDK

CLI

AWS CLI

Per ottenere le proprietà del set di immagini

Il seguente esempio di `get-image-set` codice ottiene le proprietà di un set di immagini.

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

Output:

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

Per ulteriori informazioni, consulta Ottenere le proprietà del set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta GetImageSet AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

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

       return null;
   }
```

- Per i dettagli sull'API, GetImageSetconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetImageSetReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.health_imaging_client.get_image_set(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
```

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetImageSet AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottenere i metadati del set di immagini

Usa l'`GetImageSetMetadata`azione per recuperare i metadati per un determinato set di immagini. HealthImaging I menu seguenti forniscono una procedura per gli SDK AWS Management Console e alcuni esempi di codice. AWS CLI AWS Per ulteriori informazioni, consulta `GetImageSetMetadata`l'AWS HealthImaging API Reference.

> **Note**
>
> Per impostazione predefinita, HealthImaging restituisce gli attributi dei metadati per la versione più recente di un set di immagini. Per visualizzare i metadati per una versione precedente di un set di immagini, fornisci la tua `versionId` richiesta.
> I metadati del set di immagini vengono compressi `gzip` e restituiti come oggetto JSON. Pertanto, è necessario decomprimere l'oggetto JSON prima di visualizzare i metadati normalizzati. Per ulteriori informazioni, consulta Normalizzazione dei metadati.

Per ottenere i metadati del set di immagini

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Si apre la pagina dei dettagli del Data Store e la scheda Set di immagini è selezionata per impostazione predefinita.
3. Scegliete un set di immagini.

   Viene visualizzata la pagina dei dettagli del set di immagini e i metadati del set di immagini vengono visualizzati nella sezione Visualizzatore di metadati del set di immagini.

AWS CLI e SDK

C++

SDK per C++

Funzione di utilità per ottenere i metadati dei set di immagini.

```
//! 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();
}
```

Ottieni i metadati del set di immagini senza versione.

```
        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;
        }
```

Ottieni i metadati del set di immagini con la versione.

```
        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;
        }
```

- Per i dettagli sull'API, consulta la sezione GetImageSetMetadata AWS SDK for C++API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Esempio 1: per ottenere i metadati del set di immagini senza versione

Il seguente esempio di `get-image-set-metadata` codice ottiene i metadati per un set di immagini senza specificare una versione.

Nota: `outfile` è un parametro obbligatorio

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

I metadati restituiti vengono compressi con gzip e archiviati nel file studymetadata.json.gz. Per visualizzare il contenuto dell'oggetto JSON restituito, devi prima decomprimerlo.

Output:

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

Esempio 2: per ottenere i metadati del set di immagini con la versione

Il seguente esempio di `get-image-set-metadata` codice ottiene i metadati per un set di immagini con una versione specificata.

Nota: `outfile` è un parametro obbligatorio

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

I metadati restituiti vengono compressi con gzip e archiviati nel file studymetadata.json.gz. Per visualizzare il contenuto dell'oggetto JSON restituito, devi prima decomprimerlo.

Output:

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

Per ulteriori informazioni, consulta Ottenere i metadati dei set di immagini nella Guida per gli sviluppatori.AWS HealthImaging

- Per i dettagli sull'API, consulta GetImageSetMetadata AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

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


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

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

- Per i dettagli sull'API, GetImageSetMetadataconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

Funzione di utilità per ottenere i metadati del set di immagini.

```
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;
};
```

Ottieni i metadati del set di immagini senza versione.

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

Ottieni i metadati del set di immagini con la versione.

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

- Per i dettagli sull'API, consulta la sezione GetImageSetMetadata AWS SDK for JavaScriptAPI Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

Funzione di utilità per ottenere i metadati del set di immagini.

```
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
```

Ottieni i metadati del set di immagini senza versione.

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

Ottieni i metadati del set di immagini con la versione.

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

Il codice seguente crea un'istanza dell'oggetto. MedicalImagingWrapper

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

- Per i dettagli sull'API, consulta GetImageSetMetadata AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Acquisizione dei dati dei pixel del set di immagini

Una cornice di immagine è costituita dai dati di pixel presenti all'interno di un set di immagini per creare un'immagine medica 2D. Utilizzate l'`GetImageFrame`azione per recuperare una cornice di immagine con codifica HTJ2K per un determinato set di immagini. HealthImaging I seguenti menu forniscono esempi di codice per gli SDK e. AWS CLI AWS Per ulteriori informazioni, consulta `GetImageFrame`l'AWS HealthImaging API Reference.

> **Note**
>
> Durante l'importazione, AWS HealthImaging codifica tutti i frame delle immagini in formato HTJ2K senza perdita di dati, pertanto devono essere decodificati prima di essere visualizzati in un visualizzatore di immagini. Per ulteriori informazioni, consulta Librerie di decodifica HTJ2K.

Per ottenere una cornice per l'immagine

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

> **Note**
>
> È necessario accedere ai frame delle immagini e decodificarli a livello di programmazione, poiché un visualizzatore di immagini non è disponibile in. AWS Management Console Per ulteriori informazioni sulla decodifica e la visualizzazione dei frame delle immagini, vedere. Librerie di decodifica HTJ2K

AWS CLI e SDK

C++

SDK per C++

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

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

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

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

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

        std::ofstream outfile(jphFile, std::ios::binary);
        outfile << buffer.rdbuf();
    }
    else {
```

```
        std::cout << "Error retrieving image frame." <<
 outcome.GetError().GetMessage()
                  << std::endl;

    }

    return outcome.IsSuccess();
}
```

- Per i dettagli sulle API, consulta la sezione GetImageFrame AWS SDK for C++API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per ottenere i dati dei pixel del set di immagini

Il seguente esempio di `get-image-frame` codice ottiene una cornice di immagine.

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

Nota: questo esempio di codice non include l'output perché l' GetImageFrame azione restituisce un flusso di dati di pixel al file imageframe.jph. Per informazioni sulla decodifica e la visualizzazione dei frame di immagini, vedete Librerie di decodifica HTJ2K.

Per ulteriori informazioni, consultate Ottenere i dati dei pixel del set di immagini nella Guida per gli sviluppatori.AWS HealthImaging

- Per i dettagli sull'API, consulta GetImageFrame AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

.imageFrameInformation(ImageFrameInformation.builder()

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

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,

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

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

- Per i dettagli sull'API, GetImageFrameconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

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

```
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetImageFrameReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

        :param file_path_to_write: The path to write the image frame's HTJ2K
 encoded pixel data.
        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param image_frame_id: The ID of the image frame.
        """
        try:
            image_frame = self.health_imaging_client.get_image_frame(
                datastoreId=datastore_id,
                imageSetId=image_set_id,
                imageFrameInformation={"imageFrameId": image_frame_id},
            )
            with open(file_path_to_write, "wb") as f:
                for chunk in image_frame["imageFrameBlob"].iter_chunks():
                    if chunk:
```

```
                    f.write(chunk)
        except ClientError as err:
            logger.error(
                "Couldn't get image frame. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetImageFrame AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottenere un'istanza DICOM

> **Note**
>
> L' HealthImaging `GetDICOMInstance`API è costruita in conformità allo standard DICOMWeb Retrieve (WADO-RS) per l'imaging medico basato sul web. Poiché `GetDICOMInstance` è una rappresentazione di un servizio DICOMWeb, non viene offerto tramite SDK. AWS CLI AWS

Utilizzate l'`GetDICOMInstance`azione per recuperare un'istanza DICOM (`.dcm`file) da un HealthImaging data store specificando gli UID Series, Study e Instance associati alla risorsa. È possibile specificare il set di immagini da cui recuperare una risorsa di istanza fornendo l'ID del set di

immagini come parametro di interrogazione. Inoltre, è possibile scegliere la sintassi di trasferimento per comprimere i dati DICOM, con supporto per JPEG 2000 non compresso (ELE) o High-Throughput JPEG 2000 (HTJ2K). Con`GetDICOMInstance`, è possibile interagire con sistemi che utilizzano i file binari DICOM Part 10 sfruttando contemporaneamente le interfacce native del cloud. HealthImaging

Per ottenere un'istanza DICOM (file) **`.dcm`**

1. Raccogli HealthImaging `datastoreId` e `imageSetId` parametra i valori.
2. Utilizzate l'`GetImageSetMetadata`azione con i valori `imageSetId` dei parametri `datastoreId` e per recuperare i valori dei metadati associati per `studyInstanceUID`, `seriesInstanceUID`, e. `sopInstanceUID` Per ulteriori informazioni, consulta Ottenere i metadati del set di immagini.
3. Costruisci un URL per la richiesta utilizzando i valori per`datastoreId`,, `studyInstanceUID`, `seriesInstanceUID` e. `sopInstanceUID` L'URL ha il seguente formato:

   ```
   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. Prepara e invia la tua richiesta. `GetDICOMInstance`utilizza una richiesta HTTP GET con protocollo di AWS firma Signature Version 4. Il seguente esempio di codice utilizza lo strumento da riga di `curl` comando per ottenere un'istanza DICOM (`.dcm`file) da HealthImaging.

   Shell

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

> **Note**
>
> L'`transfer-syntax`UID è facoltativo e il valore predefinito è Explicit VR Little Endian se non è incluso. Le sintassi di trasferimento supportate includono:
>
> - Explicit VR Little Endian (ELE) - `1.2.840.10008.1.2.1` (impostazione predefinita)
> - JPEG 2000 ad alta produttività con opzioni RPCL e compressione delle immagini (solo senza perdita di dati) - `1.2.840.10008.1.2.4.202`
>
> Per ulteriori informazioni, consulta Librerie di decodifica HTJ2K per AWS HealthImaging.

Modifica dei set di immagini con AWS HealthImaging

I lavori di importazione DICOM richiedono in genere la modifica dei set di immagini per i seguenti motivi:

- Sicurezza del paziente
- Coerenza dei dati
- Ridurre i costi di archiviazione

HealthImaging fornisce diverse API per semplificare il processo di modifica del set di immagini. I seguenti argomenti descrivono come modificare i set di immagini utilizzando gli AWS SDK AWS CLI and.

Argomenti

Elenco delle versioni dei set di immagini

Usa l'`ListImageSetVersions`azione per elencare la cronologia delle versioni di un set di immagini HealthImaging. I menu seguenti forniscono una procedura AWS Management Console e alcuni esempi di codice per gli AWS SDK AWS CLI and. Per ulteriori informazioni, consulta `ListImageSetVersions`l'AWS HealthImaging API Reference.

> **Note**
>
> AWS HealthImaging registra ogni modifica apportata a un set di immagini. L'aggiornamento dei metadati del set di immagini crea una nuova versione nella cronologia del set di immagini. Per ulteriori informazioni, consulta Aggiornamento dei metadati del set di immagini.

Per elencare le versioni di un set di immagini

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Si apre la pagina dei dettagli del Data Store e la scheda Set di immagini è selezionata per impostazione predefinita.
3. Scegliete un set di immagini.

   Viene visualizzata la pagina dei dettagli del set di immagini.

   La versione del set di immagini viene visualizzata nella sezione Dettagli del set di immagini.

AWS CLI e SDK

CLI

AWS CLI

Per elencare le versioni dei set di immagini

Il seguente esempio di `list-image-set-versions` codice elenca la cronologia delle versioni di un set di immagini.

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

Output:

```
{
    "imageSetPropertiesList": [
        {
            "ImageSetWorkflowStatus": "UPDATED",
            "versionId": "4",
            "updatedAt": 1680029436.304,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
```

```
            "ImageSetWorkflowStatus": "UPDATED",
            "versionId": "3",
            "updatedAt": 1680029163.325,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
            "ImageSetWorkflowStatus": "COPY_FAILED",
            "versionId": "2",
            "updatedAt": 1680027455.944,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "message": "INVALID_REQUEST:  Series of SourceImageSet and
 DestinationImageSet don't match.",
            "createdAt": 1680027126.436
        },
        {
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "versionId": "1",
            "ImageSetWorkflowStatus": "COPIED",
            "createdAt": 1680027126.436
        }
    ]
}
```

Per ulteriori informazioni, consultate Elenco delle versioni dei set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta ListImageSetVersions AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

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

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

        return null;
    }
```

- Per i dettagli sull'API, ListImageSetVersionsconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListImageSetVersionsReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListImageSetVersions AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiornamento dei metadati del set di immagini

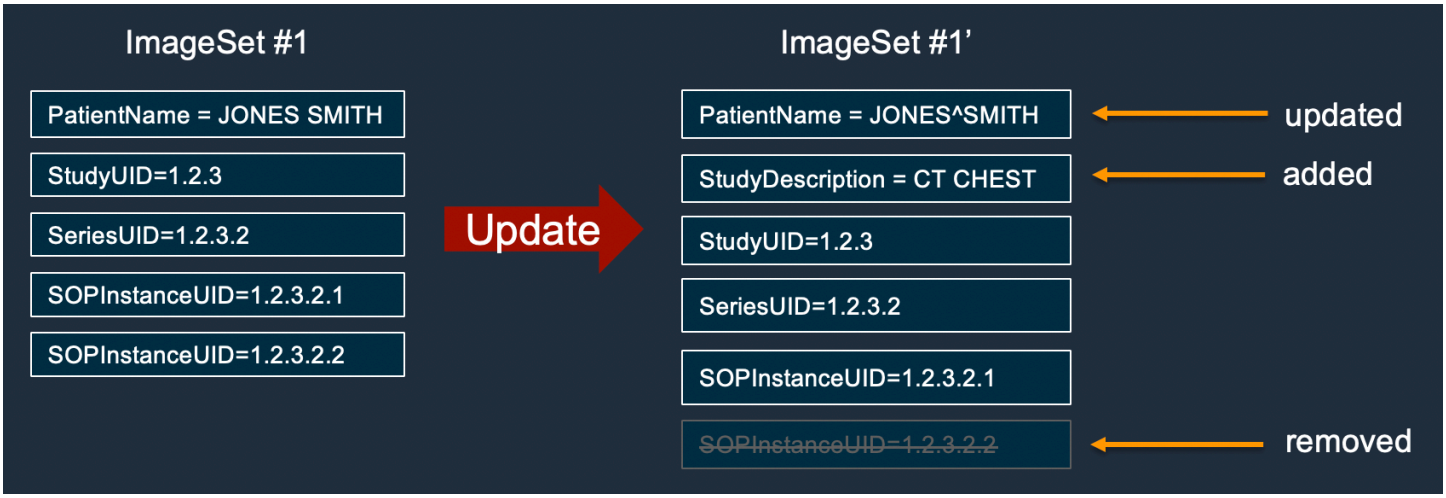
Usa l'`UpdateImageSetMetadata`azione per aggiornare i metadati dei set di immagini in AWS HealthImaging. È possibile utilizzare questo processo asincrono per aggiungere, aggiornare e rimuovere gli attributi dei metadati del set di immagini, che sono manifestazioni degli elementi di normalizzazione DICOM creati durante l'importazione. Questa `UpdateImageSetMetadata` azione consente anche di rimuovere le istanze Series e SOP per mantenere i set di immagini sincronizzati con i sistemi esterni e rendere anonimi i metadati dei set di immagini. Per ulteriori informazioni, `UpdateImageSetMetadata`consulta AWS HealthImaging API Reference.

Comprendere **`UpdateImageSetMetadata`**

> **Note**
>
> Le importazioni DICOM nel mondo reale richiedono l'aggiornamento, l'aggiunta e la rimozione di attributi dai metadati del set di immagini. Tieni presente i seguenti punti quando aggiorni i metadati del set di immagini:
>
> - L'aggiornamento dei metadati del set di immagini crea una nuova versione nella cronologia del set di immagini. Per ulteriori informazioni, consulta Elenco delle versioni dei set di immagini.

- L'aggiornamento dei metadati del set di immagini è un processo asincrono. Pertanto, `imageSetState`sono disponibili elementi di `imageSetWorkflowStatus`risposta che forniscono il rispettivo stato e lo stato di un set di immagini bloccato. Non è possibile eseguire altre operazioni di scrittura su un set di immagini bloccato.
- I vincoli degli elementi DICOM vengono applicati agli aggiornamenti dei metadati. Per ulteriori informazioni, consulta vincoli dei metadati DICOM.
- Se un'azione di aggiornamento dei metadati del set di immagini non ha esito positivo, richiama ed esamina l'elemento di risposta. `message`

Il diagramma seguente rappresenta i metadati del set di immagini in corso di aggiornamento in. HealthImaging



Per aggiornare i metadati del set di immagini

Scegli una scheda in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS CLI e SDK

CLI

AWS CLI

Per inserire o aggiornare un attributo nei metadati del set di immagini

Il seguente esempio di `update-image-set-metadata` codice inserisce o aggiorna un attributo nei metadati del set di immagini.

```
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
```

Contenuto di `metadata-updates.json`

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

Nota: `updatableAttributes` è una stringa JSON con codifica Base64. Ecco la stringa JSON non codificata.

{» SchemaVersion «:1.1, "Paziente»: {"DICOM»: {» «:"MX^MX"PatientName}}}

Output:

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

Per rimuovere un attributo dai metadati del set di immagini

Il seguente esempio di `update-image-set-metadata` codice rimuove un attributo dai metadati del set di immagini.

```
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
```

Contenuto di `metadata-updates.json`

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

Nota: `removableAttributes` è una stringa JSON con codifica Base64. Ecco la stringa JSON non codificata. La chiave e il valore devono corrispondere all'attributo da rimuovere.

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

Output:

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

Per rimuovere un'istanza dai metadati del set di immagini

Il seguente esempio di `update-image-set-metadata` codice rimuove un'istanza dai metadati del set di immagini.

```
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
```

Contenuto di `metadata-updates.json`

```
{
```

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

Nota: `removableAttributes` è una stringa JSON con codifica Base64. Ecco la stringa JSON non codificata.

{"1.1.1.1.1.12345.123456789012.123.12345678901234.1": {" Istanze»: {"1.1.1.1.1.12345.123456789012.123.1234567890123.1234567890123.12345678901234.1": {}}}}}

Output:

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

Per ulteriori informazioni, consulta Aggiornamento dei metadati del set di immagini nella Guida per gli sviluppatori.AWS HealthImaging

- Per i dettagli sull'API, consulta UpdateImageSetMetadata AWS CLICommand Reference.

Java

SDK per Java 2.x

```
   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);
        }
    }
```

Caso d'uso #1: Inserimento o aggiornamento di un attributo.

```
            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);
```

Caso d'uso #2: rimuovere un attributo.

```
            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);
```

Caso d'uso #3: rimuove un'istanza.

```
            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);
```

- Per i dettagli sull'API, consulta la UpdateImageSetMetadatasezione AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 version.
 * @param {{}} updateMetadata - The metadata to update.
 */
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;
};
```

Caso d'uso #1: inserire o aggiornare un attributo.

```
        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);
```

Caso d'uso #2: rimuovere un attributo.

```
        // 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);
```

Caso d'uso #3: rimuove un'istanza.

```
        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);
```

- Per i dettagli sull'API, consulta la UpdateImageSetMetadatasezione AWS SDK for JavaScript API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

Caso d'uso #1: Inserimento o aggiornamento di un attributo.

```
            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
        )
```

Caso d'uso #2: rimuovere un attributo.

```
        # 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
        )
```

Caso d'uso #3: rimuove un'istanza.

```
        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
        )
```

- Per i dettagli sull'API, consulta UpdateImageSetMetadata AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Copiare un set di immagini

Usa l'`CopyImageSet`azione per copiare un set di immagini. HealthImaging Utilizzate questo processo asincrono per copiare il contenuto di un set di immagini in un set di immagini nuovo o esistente. È possibile copiare in una nuova immagine per dividere un set di immagini e creare una copia separata. Potete anche copiare in un set di immagini esistente per unire due set di immagini. Per ulteriori informazioni, `CopyImageSet`consulta AWS HealthImaging API Reference.

Comprendere **`CopyImageSet`**

> **Note**
>
> Tieni a mente i seguenti punti quando copi un set di immagini:
>
> - La copia di un set di immagini crea una nuova versione nella cronologia del set di immagini. Per ulteriori informazioni, consulta Elenco delle versioni dei set di immagini.
> - La copia di un set di immagini è un processo asincrono. Pertanto, gli elementi di risposta state (`imageSetState`) e status (`imageSetWorkflowStatus`) sono disponibili per indicare l'operazione in corso su un set di immagini bloccato. Non è possibile eseguire altre operazioni di scrittura su un set di immagini bloccato.

- `CopyImageSet`richiede UID di istanza SOP univoci per avere successo. Pertanto, è necessario scegliere l'istanza SOP corretta rimuovendola dal set di immagini indesiderato.
- Se un'azione di copia del set di immagini non ha esito positivo, chiamate `GetImageSet` ed esaminate la `message`proprietà. Per ulteriori informazioni, consulta Ottenere le proprietà del set di immagini.
- Le importazioni DICOM nel mondo reale possono generare più set di immagini per serie DICOM. Considerate i seguenti punti quando utilizzate l'azione: `CopyImageSet`
  - Copia le istanze da un set di immagini a un altro
  - Copy richiede che entrambi i set di immagini abbiano metadati coerenti

Per copiare un set di immagini

Scegli una scheda in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS CLI e SDK

CLI

AWS CLI

Esempio 1: copiare un set di immagini senza una destinazione.

Il seguente esempio di `copy-image-set` codice crea una copia duplicata di un set di immagini senza una destinazione.

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

Output:

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042357.432,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
```

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

Esempio 2: Copiare un set di immagini con una destinazione.

Il seguente esempio di `copy-image-set` codice crea una copia duplicata di un set di immagini con una destinazione.

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "1" },
 "destinationImageSet": { "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
 "latestVersionId": "1"} }'
```

Output:

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042505.135,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042505.135,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
```

```
    },
    "datastoreId": "12345678901234567890123456789012"
}
```

Per ulteriori informazioni, consultate Copiare un set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta CopyImageSet AWS CLICommand Reference.

Java

SDK per Java 2.x

```
   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 "";
    }
```

- Per i dettagli sull'API, CopyImageSetconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

Funzione di utilità per copiare un set di immagini.

```
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: 'xxxxxxxxxxxxx',
  //             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;
};
```

Copia un set di immagini senza una destinazione.

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

Copia un set di immagini con una destinazione.

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

- Per i dettagli sull'API, consulta la CopyImageSetsezione AWS SDK for JavaScript API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

Funzione di utilità per copiare un set di immagini.

```
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"]
```

Copia un set di immagini senza una destinazione.

```
            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,
            )
```

Copia un set di immagini con una destinazione.

```
            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,
            )
```

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta CopyImageSet AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminazione di un set di immagini

Usa l'`DeleteImageSet`azione per eliminare un set di immagini. HealthImaging I menu seguenti forniscono una procedura AWS Management Console e alcuni esempi di codice per gli AWS SDK AWS CLI and. Per ulteriori informazioni, consulta `DeleteImageSet`l'AWS HealthImaging API Reference.

Per eliminare un set di immagini

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.

2. Scegli un archivio dati.

   Si apre la pagina dei dettagli del Data Store e la scheda Set di immagini è selezionata per impostazione predefinita.

3. Scegliete un set di immagini e scegliete Elimina.

   Si apre la finestra modale Elimina set di immagini.

4. Fornite l'ID del set di immagini e scegliete Elimina set di immagini.

AWS CLI e SDK

C++

SDK per C++

```
//! Routine which deletes an AWS HealthImaging image set.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param imageSetID: The image set ID.
  \param clientConfig: Aws client configuration.
  \return bool: Function succeeded.
  */
bool AwsDoc::Medical_Imaging::deleteImageSet(
        const Aws::String &dataStoreID, const Aws::String &imageSetID,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::MedicalImaging::MedicalImagingClient client(clientConfig);
    Aws::MedicalImaging::Model::DeleteImageSetRequest request;
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);
    Aws::MedicalImaging::Model::DeleteImageSetOutcome outcome =
 client.DeleteImageSet(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted image set " << imageSetID
                  << " from data store " << dataStoreID << std::endl;
    }
    else {
        std::cerr << "Error deleting image set " << imageSetID << " from data
 store "
                  << dataStoreID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
```

```
    }

    return outcome.IsSuccess();
}
```

- Per i dettagli sulle API, consulta la sezione DeleteImageSet AWS SDK for C++API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per eliminare un set di immagini

Il seguente esempio di `delete-image-set` codice elimina un set di immagini.

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

Output:

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

Per ulteriori informazioni, consultate Eliminazione di un set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta DeleteImageSet AWS CLICommand Reference.

Java

SDK per Java 2.x

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

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- Per i dettagli sull'API, DeleteImageSetconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
```

```
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteImageSetReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta DeleteImageSet AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Taggare le risorse con AWS HealthImaging

Puoi assegnare metadati HealthImaging alle risorse AWS (archivi di dati e set di immagini) sotto forma di tag. Ogni tag è un'etichetta composta da una chiave e da un valore definiti dall'utente. I tag ti aiutano a gestire, identificare, organizzare, cercare e filtrare le risorse.

> **Importante**
>
> Non archiviate nei tag informazioni sanitarie protette (PHI), informazioni di identificazione personale (PII) o altre informazioni riservate o sensibili. I tag non sono destinati ad essere utilizzati per dati privati o sensibili.

I seguenti argomenti descrivono come utilizzare le operazioni di HealthImaging tagging utilizzando gli SDK, e. AWS Management Console AWS CLI AWS Per ulteriori informazioni, consulta Taggare le AWS risorse nella Guida.Riferimenti generali di AWS

Argomenti

- Taggare una risorsa
- Elencare i tag per una risorsa
- Rimuovere il tag di una risorsa

Taggare una risorsa

Usa l'`TagResource`azione per etichettare una risorsa in AWS HealthImaging. I seguenti esempi di codice descrivono come utilizzare l'`TagResource`azione con AWS Management Console AWS CLI, e AWS SDK. Per ulteriori informazioni, consulta Taggare le AWS risorse nella Riferimenti generali di AWS Guida.

Per etichettare una risorsa (archivio dati)

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.

2. Scegli un archivio dati.

   Viene visualizzata la pagina dei dettagli del Data store.
3. Seleziona la scheda Details (Dettagli).
4. Nella sezione Tag, scegli Gestisci tag.

   Si apre la pagina Gestisci tag.
5. Scegli Aggiungi nuovo tag.
6. Inserisci una chiave e un valore (opzionale).
7. Seleziona Salvataggio delle modifiche.

AWS CLI e SDK

CLI

AWS CLI

Esempio 1: etichettare un archivio dati

I seguenti esempi di `tag-resource` codice contrassegnano un data store.

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

Questo comando non produce alcun output.

Esempio 2: etichettare un set di immagini

I seguenti esempi di `tag-resource` codice contrassegnano un set di immagini.

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

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Tagging resources with AWS HealthImaging nella AWS HealthImaging Developer Guide.

- Per i dettagli sull'API, consulta TagResource AWS CLICommand Reference.

Java

SDK per Java 2.x

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

           medicalImagingClient.tagResource(tagResourceRequest);

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

- Per i dettagli sull'API, TagResourceconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API TagResourceReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta TagResource AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elencare i tag per una risorsa

Usa l'`ListTagsForResource`azione per elencare i tag per una risorsa in AWS HealthImaging. I seguenti esempi di codice descrivono come utilizzare l'`ListTagsForResource`azione con AWS Management Console AWS CLI, e AWS SDK. Per ulteriori informazioni, consulta Taggare le AWS risorse nella Riferimenti generali di AWS Guida.

Per elencare i tag di una risorsa (archivio dati)

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

   Viene visualizzata la pagina dei dettagli del Data store.
3. Seleziona la scheda Details (Dettagli).

   Nella sezione Tag, sono elencati tutti i tag del data store.

AWS CLI e SDK

CLI

AWS CLI

Esempio 1: elencare i tag delle risorse per un archivio dati

Il seguente esempio di `list-tags-for-resource` codice elenca i tag per un data store.

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

Output:

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

```
    }
}
```

Esempio 2: Elencare i tag delle risorse per un set di immagini

Il seguente esempio di `list-tags-for-resource` codice elenca i tag per un set di immagini.

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

Output:

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

Per ulteriori informazioni, consulta Tagging resources with AWS HealthImaging nella AWS HealthImaging Developer Guide.

- Per i dettagli sull'API, consulta ListTagsForResource AWS CLICommand Reference.

Java

SDK per Java 2.x

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

           return
medicalImagingClient.listTagsForResource(listTagsForResourceRequest);
```

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

        return null;
    }
```

- Per i dettagli sull'API, ListTagsForResourceconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

```
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListTagsForResourceReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListTagsForResource AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Rimuovere il tag di una risorsa

Usa l'`UntagResource`azione per rimuovere i tag da una risorsa in AWS HealthImaging. I seguenti esempi di codice descrivono come utilizzare l'`UntagResource`azione con AWS Management Console AWS CLI, e AWS SDK. Per ulteriori informazioni, consulta Taggare le AWS risorse nella Riferimenti generali di AWS Guida.

Per rimuovere i tag da una risorsa (archivio dati)

Scegli un menu in base alle tue preferenze di accesso ad AWS HealthImaging.

AWS Console

1. Apri la pagina degli archivi dati della HealthImaging console.
2. Scegli un archivio dati.

Viene visualizzata la pagina dei dettagli del Data store.

3. Seleziona la scheda Details (Dettagli).
4. Nella sezione Tag, scegli Gestisci tag.

   Si apre la pagina Gestisci tag.
5. Scegli Rimuovi accanto al tag che desideri rimuovere.
6. Seleziona Salvataggio delle modifiche.

AWS CLI e SDK

CLI

AWS CLI

Esempio 1: rimuovere i tag da un archivio dati

Il seguente esempio di `untag-resource` codice rimuove i tag da un data store.

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

Questo comando non produce alcun output.

Esempio 2: rimuovere i tag da un set di immagini

Il seguente esempio di `untag-resource` codice rimuove i tag da un set di immagini.

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

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Tagging resources with AWS HealthImaging nella Developer Guide.AWS HealthImaging

- Per i dettagli sull'API, consulta UntagResource AWS CLICommand Reference.

Java

SDK per Java 2.x

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

           medicalImagingClient.untagResource(untagResourceRequest);

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

- Per i dettagli sull'API, UntagResourceconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [UntagResource](#)Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta UntagResource AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esempi di codice per l' HealthImaging utilizzo degli AWS SDK

I seguenti esempi di codice mostrano come utilizzare un kit HealthImaging di sviluppo AWS software (SDK).

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo HealthImaging con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Nozioni di base

Salve HealthImaging

I seguenti esempi di codice mostrano come iniziare a utilizzare HealthImaging.

C++

SDK per C++

Codice per il file CMake C MakeLists .txt.

```
# 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})
```

Codice per il file sorgente hello_health_imaging.cpp.

```
#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;
}
```

- Per i dettagli sull'API, consulta la ListDatastoressezione AWS SDK for C++ API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

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

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListDatastoresReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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

logger = logging.getLogger(__name__)
```

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

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


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

- Per i dettagli sull'API, consulta ListDatastores AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esempi di codice

Azioni per l' HealthImaging utilizzo degli AWS SDK

I seguenti esempi di codice mostrano come eseguire HealthImaging azioni individuali con gli AWS SDK. Questi estratti richiamano l' HealthImaging API e sono estratti di codice di programmi più grandi che devono essere eseguiti nel contesto. Ogni esempio include un collegamento a GitHub, dove è possibile trovare le istruzioni per la configurazione e l'esecuzione del codice.

Gli esempi seguenti includono solo le operazioni più comunemente utilizzate. Per un elenco completo, consulta la Documentazione di riferimento delle API AWS HealthImaging.

Esempi

- Utilizzo CopyImageSet con un AWS SDK o una CLI
- Utilizzo CreateDatastore con un AWS SDK o una CLI
- Utilizzo DeleteDatastore con un AWS SDK o una CLI
- Utilizzo DeleteImageSet con un AWS SDK o una CLI
- Utilizzo GetDICOMImportJob con un AWS SDK o una CLI
- Utilizzo GetDatastore con un AWS SDK o una CLI
- Utilizzo GetImageFrame con un AWS SDK o una CLI
- Utilizzo GetImageSet con un AWS SDK o una CLI
- Utilizzo GetImageSetMetadata con un AWS SDK o una CLI
- Utilizzo ListDICOMImportJobs con un AWS SDK o una CLI
- Utilizzo ListDatastores con un AWS SDK o una CLI
- Utilizzo ListImageSetVersions con un AWS SDK o una CLI
- Utilizzo ListTagsForResource con un AWS SDK o una CLI
- Utilizzo SearchImageSets con un AWS SDK o una CLI
- Utilizzo StartDICOMImportJob con un AWS SDK o una CLI
- Utilizzo TagResource con un AWS SDK o una CLI
- Utilizzo UntagResource con un AWS SDK o una CLI
- Utilizzo UpdateImageSetMetadata con un AWS SDK o una CLI

Utilizzo **`CopyImageSet`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`CopyImageSet`.

CLI

AWS CLI

Esempio 1: copiare un set di immagini senza una destinazione.

Il seguente esempio di `copy-image-set` codice crea una copia duplicata di un set di immagini senza una destinazione.

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

Output:

```
{
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042357.432,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042357.432,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
}
```

Esempio 2: Copiare un set di immagini con una destinazione.

Il seguente esempio di `copy-image-set` codice crea una copia duplicata di un set di immagini con una destinazione.

```
aws medical-imaging copy-image-set \
    --datastore-id 12345678901234567890123456789012 \
    --source-image-set-id ea92b0d8838c72a3f25d00d13616f87e \
    --copy-image-set-information '{"sourceImageSet": {"latestVersionId": "1" },
 "destinationImageSet": { "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
 "latestVersionId": "1"} }'
```

Output:

```
{
```

```
    "destinationImageSetProperties": {
        "latestVersionId": "2",
        "imageSetWorkflowStatus": "COPYING",
        "updatedAt": 1680042505.135,
        "imageSetId": "b9a06fef182a5f992842f77f8e0868e5",
        "imageSetState": "LOCKED",
        "createdAt": 1680042357.432
    },
    "sourceImageSetProperties": {
        "latestVersionId": "1",
        "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS",
        "updatedAt": 1680042505.135,
        "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
        "imageSetState": "LOCKED",
        "createdAt": 1680027126.436
    },
    "datastoreId": "12345678901234567890123456789012"
}
```

Per ulteriori informazioni, consultate Copiare un set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta CopyImageSet AWS CLICommand Reference.

Java

SDK per Java 2.x

```
   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 "";
    }
```

- Per i dettagli sull'API, CopyImageSetconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

Funzione di utilità per copiare un set di immagini.

```
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;
};
```

Copia un set di immagini senza una destinazione.

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

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

Copia un set di immagini con una destinazione.

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

- Per i dettagli sull'API, consulta la CopyImageSetsezione AWS SDK for JavaScript API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

Funzione di utilità per copiare un set di immagini.

```
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"]
```

Copia un set di immagini senza una destinazione.

```
            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,
            )
```

Copia un set di immagini con una destinazione.

```
            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,
            )
```

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta CopyImageSet AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`CreateDatastore`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`CreateDatastore`.

Bash

AWS CLI con lo script Bash

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

- Per i dettagli sull'API, vedere CreateDatastorein AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per creare un archivio dati

Il seguente esempio di `create-datastore` codice crea un archivio dati con il nome`my-datastore`.

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

Output:

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

Per ulteriori informazioni, consulta Creazione di un data store nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta CreateDatastore AWS CLICommand Reference.

Java

SDK per Java 2.x

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

       return "";
   }
```

- Per i dettagli sull'API, CreateDatastoreconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API CreateDatastoreReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta CreateDatastore AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **DeleteDatastore** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`DeleteDatastore`.

Bash

AWS CLI con lo script Bash

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

- Per i dettagli sull'API, vedere DeleteDatastorein AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per eliminare un archivio dati

Il seguente esempio di `delete-datastore` codice elimina un data store.

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

Output:

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

Per ulteriori informazioni, consulta Eliminazione di un data store nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta DeleteDatastore AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

- Per i dettagli sull'API, DeleteDatastoreconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

```
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteDatastoreReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta DeleteDatastore AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteImageSet`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`DeleteImageSet`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. Puoi vedere questa azione nel contesto nel seguente esempio di codice:

- Inizia con set di immagini e cornici di immagini

C++

SDK per C++

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

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

    return outcome.IsSuccess();
}
```

- Per i dettagli sull'API, consulta DeleteImageSet AWS SDK for C++API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per eliminare un set di immagini

Il seguente esempio di `delete-image-set` codice elimina un set di immagini.

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

Output:

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

Per ulteriori informazioni, consultate Eliminazione di un set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta DeleteImageSet AWS CLICommand Reference.

Java

SDK per Java 2.x

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

           medicalImagingClient.deleteImageSet(deleteImageSetRequest);

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

- Per i dettagli sull'API, DeleteImageSetconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteImageSetReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

```
            return delete_results
```

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta DeleteImageSet AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetDICOMImportJob`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetDICOMImportJob`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. Puoi vedere questa azione nel contesto nel seguente esempio di codice:

- Inizia con set di immagini e cornici di immagini

C++

SDK per C++

```
//! Routine which gets a HealthImaging DICOM import job's properties.
/*!
  \param dataStoreID: The HealthImaging data store ID.
  \param importJobID: The DICOM import job ID
```

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

    return outcome;
}
```

- Per i dettagli sull'API, consulta GetDicom ImportJob in AWS SDK for C++ API Reference.

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per ottenere le proprietà di un processo di importazione dicom

Il seguente esempio di `get-dicom-import-job` codice ottiene le proprietà di un processo di importazione dicom.

```
aws medical-imaging get-dicom-import-job \
```

```
    --datastore-id "12345678901234567890123456789012" \
    --job-id "09876543210987654321098765432109"
```

Output:

```
{
    "jobProperties": {
        "jobId": "09876543210987654321098765432109",
        "jobName": "my-job",
        "jobStatus": "COMPLETED",
        "datastoreId": "12345678901234567890123456789012",
        "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
        "endedAt": "2022-08-12T11:29:42.285000+00:00",
        "submittedAt": "2022-08-12T11:28:11.152000+00:00",
        "inputS3Uri": "s3://medical-imaging-dicom-input/dicom_input/",
        "outputS3Uri": "s3://medical-imaging-output/
job_output/12345678901234567890123456789012-
DicomImport-09876543210987654321098765432109/"
    }
}
```

Per ulteriori informazioni, consulta Ottenere le proprietà del processo di importazione nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta GetDicom ImportJob in AWS CLI Command Reference.

Java

SDK per Java 2.x

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

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

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

        return null;
    }
```

- Per i dettagli sull'API, consulta getDicom ImportJob in API Reference.AWS SDK for Java 2.x

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "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: '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-'xxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta getDICOM ImportJob in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetDICOM ImportJob nella guida di riferimento all'API AWS SDK for Python (Boto3).

> **Note**
>
> C' GitHubè di più su. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetDatastore`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetDatastore`.

Bash

AWS CLI con lo script Bash

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

- Per i dettagli sull'API, vedere GetDatastorein AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per ottenere le proprietà di un archivio dati

Il seguente esempio di `get-datastore` codice ottiene le proprietà di un data store.

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

Output:

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

Per ulteriori informazioni, consulta Ottenere le proprietà del data store nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta GetDatastore AWS CLICommand Reference.

Java

SDK per Java 2.x

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

       return null;
   }
```

- Per i dettagli sull'API, GetDatastoreconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetDatastoreReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetDatastore AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetImageFrame`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetImageFrame`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. Puoi vedere questa azione nel contesto nel seguente esempio di codice:

- Inizia con set di immagini e cornici di immagini

C++

SDK per C++

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

    Aws::MedicalImaging::Model::GetImageFrameRequest request;
```

```
    request.SetDatastoreId(dataStoreID);
    request.SetImageSetId(imageSetID);

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

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

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

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

    }

    return outcome.IsSuccess();
}
```

- Per i dettagli sull'API, consulta GetImageFrame AWS SDK for C++API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per ottenere i dati dei pixel del set di immagini

Il seguente esempio di `get-image-frame` codice ottiene una cornice di immagine.

```
aws medical-imaging get-image-frame \
    --datastore-id "1234567890123456789012345678901 2" \
    --image-set-id "98765412345612345678907890789012" \
    --image-frame-information imageFrameId=3abf5d5d7ae72f80a0ec81b2c0de3ef4 \
    imageframe.jph
```

Nota: questo esempio di codice non include l'output perché l' GetImageFrame azione restituisce un flusso di dati di pixel al file imageframe.jph. Per informazioni sulla decodifica e la visualizzazione dei frame di immagini, vedete Librerie di decodifica HTJ2K.

Per ulteriori informazioni, consultate Ottenere i dati dei pixel del set di immagini nella Guida per gli sviluppatori.AWS HealthImaging

- Per i dettagli sull'API, consulta GetImageFrame AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

.imageFrameInformation(ImageFrameInformation.builder()

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

medicalImagingClient.getImageFrame(getImageSetMetadataRequest,
```

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

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

- Per i dettagli sull'API, GetImageFrameconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-
encoded image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
```

```
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

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

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetImageFrameReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetImageFrame AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetImageSet`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetImageSet`.

CLI

AWS CLI

Per ottenere le proprietà del set di immagini

Il seguente esempio di `get-image-set` codice ottiene le proprietà di un set di immagini.

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

Output:

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

Per ulteriori informazioni, consulta Ottenere le proprietà del set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta GetImageSet AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

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

       return null;
   }
```

- Per i dettagli sull'API, GetImageSetconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetImageSetReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :param image_set_id: The ID of the image set.
        :param version_id: The optional version of the image set.
        :return: The image set properties.
        """
        try:
            if version_id:
                image_set = self.health_imaging_client.get_image_set(
                    imageSetId=image_set_id,
                    datastoreId=datastore_id,
                    versionId=version_id,
                )
            else:
```

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta GetImageSet AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetImageSetMetadata`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetImageSetMetadata`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. Puoi vedere questa azione nel contesto nel seguente esempio di codice:

- Inizia con set di immagini e cornici di immagini

C++

SDK per C++

Funzione di utilità per ottenere i metadati del set di immagini.

```
//! 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();
```

```
}
```

Ottieni i metadati del set di immagini senza versione.

```
        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;
        }
```

Ottieni i metadati del set di immagini con la versione.

```
        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;
        }
```

- Per i dettagli sull'API, consulta la sezione GetImageSetMetadata AWS SDK for C++API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Esempio 1: per ottenere i metadati del set di immagini senza versione

Il seguente esempio di `get-image-set-metadata` codice ottiene i metadati per un set di immagini senza specificare una versione.

Nota: `outfile` è un parametro obbligatorio

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

I metadati restituiti vengono compressi con gzip e memorizzati nel file studymetadata.json.gz. Per visualizzare il contenuto dell'oggetto JSON restituito, devi prima decomprimerlo.

Output:

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

Esempio 2: per ottenere i metadati del set di immagini con la versione

Il seguente esempio di `get-image-set-metadata` codice ottiene i metadati per un set di immagini con una versione specificata.

Nota: `outfile` è un parametro obbligatorio

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

I metadati restituiti vengono compressi con gzip e memorizzati nel file studymetadata.json.gz. Per visualizzare il contenuto dell'oggetto JSON restituito, devi prima decomprimerlo.

Output:

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

Per ulteriori informazioni, consulta Ottenere i metadati dei set di immagini nella Guida per gli sviluppatori.AWS HealthImaging

- Per i dettagli sull'API, consulta GetImageSetMetadata AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

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


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

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

- Per i dettagli sull'API, GetImageSetMetadataconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

Funzione di utilità per ottenere i metadati del set di immagini.

```
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;
};
```

Ottieni i metadati del set di immagini senza versione.

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

Ottieni i metadati del set di immagini con la versione.

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

- Per i dettagli sull'API, consulta la sezione GetImageSetMetadata AWS SDK for JavaScriptAPI Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

Funzione di utilità per ottenere i metadati del set di immagini.

```
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
```

Ottieni i metadati del set di immagini senza versione.

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

Ottieni i metadati del set di immagini con la versione.

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

Il codice seguente crea un'istanza dell'oggetto. MedicalImagingWrapper

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

- Per i dettagli sull'API, consulta GetImageSetMetadata AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListDICOMImportJobs`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`ListDICOMImportJobs`.

CLI

AWS CLI

Per elencare i lavori di importazione dicom

Il seguente esempio di `list-dicom-import-jobs` codice elenca i processi di importazione dicom.

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

Output:

```
{
    "jobSummaries": [
        {
            "jobId": "09876543210987654321098765432109",
            "jobName": "my-job",
```

```
            "jobStatus": "COMPLETED",
            "datastoreId": "12345678901234567890123456789012",
            "dataAccessRoleArn": "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole",
            "endedAt": "2022-08-12T11:21:56.504000+00:00",
            "submittedAt": "2022-08-12T11:20:21.734000+00:00"
        }
    ]
}
```

Per ulteriori informazioni, consulta Elencare i lavori di importazione nella AWS HealthImaging Developer Guide.

- Per i dettagli sull'API, consulta ListDicom ImportJobs in AWS CLI Command Reference.

Java

SDK per Java 2.x

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

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

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

- Per i dettagli sull'API, consulta ListDicom ImportJobs in API Reference.AWS SDK for Java 2.x

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta ListDicom ImportJobs in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

        :param datastore_id: The ID of the data store.
        :return: The list of jobs.
```

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListDicom ImportJobs in AWS SDK for Python (Boto3) API Reference.

> **Note**
>
> C' GitHubè di più su. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListDatastores`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`ListDatastores`.

Bash

AWS CLI con lo script Bash

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

- Per i dettagli sull'API, vedere ListDatastoresin AWS CLI Command Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per elencare gli archivi di dati

Il seguente esempio di `list-datastores` codice elenca gli archivi dati disponibili.

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

Output:

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

Per ulteriori informazioni, consulta Elencare gli archivi di dati nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta ListDatastores AWS CLICommand Reference.

Java

SDK per Java 2.x

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

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

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

        return null;
    }
```

- Per i dettagli sull'API, ListDatastoresconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
```

```
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListDatastoresReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListDatastores AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListImageSetVersions`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`ListImageSetVersions`.

CLI

AWS CLI

Per elencare le versioni dei set di immagini

Il seguente esempio di `list-image-set-versions` codice elenca la cronologia delle versioni di un set di immagini.

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

Output:

```
{
    "imageSetPropertiesList": [
        {
            "ImageSetWorkflowStatus": "UPDATED",
            "versionId": "4",
            "updatedAt": 1680029436.304,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
            "ImageSetWorkflowStatus": "UPDATED",
            "versionId": "3",
```

```
            "updatedAt": 1680029163.325,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "createdAt": 1680027126.436
        },
        {
            "ImageSetWorkflowStatus": "COPY_FAILED",
            "versionId": "2",
            "updatedAt": 1680027455.944,
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "message": "INVALID_REQUEST:  Series of SourceImageSet and
 DestinationImageSet don't match.",
            "createdAt": 1680027126.436
        },
        {
            "imageSetId": "ea92b0d8838c72a3f25d00d13616f87e",
            "imageSetState": "ACTIVE",
            "versionId": "1",
            "ImageSetWorkflowStatus": "COPIED",
            "createdAt": 1680027126.436
        }
    ]
}
```

Per ulteriori informazioni, consultate Elenco delle versioni dei set di immagini nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta ListImageSetVersions AWS CLICommand Reference.

Java

SDK per Java 2.x

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

```
                    .build();

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

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

        return null;
    }
```

- Per i dettagli sull'API, ListImageSetVersionsconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListImageSetVersionsReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListImageSetVersions AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo `ListTagsForResource` con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`ListTagsForResource`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Taggare un archivio dati
- Etichettare un set di immagini

CLI

AWS CLI

Esempio 1: per elencare i tag delle risorse per un archivio dati

Il seguente esempio di `list-tags-for-resource` codice elenca i tag per un data store.

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

Output:

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

Esempio 2: Elencare i tag delle risorse per un set di immagini

Il seguente esempio di `list-tags-for-resource` codice elenca i tag per un set di immagini.

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

Output:

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

Per ulteriori informazioni, consulta Tagging resources with AWS HealthImaging nella AWS HealthImaging Developer Guide.

- Per i dettagli sull'API, consulta ListTagsForResource AWS CLICommand Reference.

Java

SDK per Java 2.x

```
    public static ListTagsForResourceResponse
 listMedicalImagingResourceTags(MedicalImagingClient medicalImagingClient,
```

```
            String resourceArn) {
        try {
            ListTagsForResourceRequest listTagsForResourceRequest =
 ListTagsForResourceRequest.builder()
                    .resourceArn(resourceArn)
                    .build();

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

        return null;
    }
```

- Per i dettagli sull'API, ListTagsForResourceconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data
 store or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/
ghi"
) => {
  const response = await medicalImagingClient.send(
```

```
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListTagsForResourceReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

        :param resource_arn: The ARN of the resource.
```

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta ListTagsForResource AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`SearchImageSets`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`SearchImageSets`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. Puoi vedere questa azione nel contesto nel seguente esempio di codice:

- Inizia con set di immagini e cornici di immagini

C++

SDK per C++

La funzione di utilità per la ricerca di set di immagini.

```
//! 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;
}
```

Caso d'uso #1: operatore EQUAL.

```
        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;
            }
        }
```

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

```
         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;
```

```
            }
        }
```

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano precedentemente persistenti.

```
        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;
            }
        }
```

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
        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;
            }
        }
```

- Per i dettagli sull'API, vedere in API Reference. SearchImageSetsAWS SDK for C++

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Esempio 1: per cercare set di immagini con un operatore EQUAL

Il seguente esempio di `search-image-sets` codice utilizza l'operatore EQUAL per cercare set di immagini in base a un valore specifico.

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

Contenuto di `search-criteria.json`

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

Output:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
```

```
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
             "DICOMPatientSex": "F",
             "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
             "DICOMPatientBirthDate": "19201120",
             "DICOMStudyDescription": "UNKNOWN",
             "DICOMPatientId": "SUBJECT08701",
             "DICOMPatientName": "Melissa844 Huel628",
             "DICOMNumberOfStudyRelatedInstances": 1,
             "DICOMStudyTime": "140728",
             "DICOMNumberOfStudyRelatedSeries": 1
            },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

Esempio 2: per cercare set di immagini con un operatore BETWEEN utilizzando DICOM StudyDate e DICOM StudyTime

Il seguente esempio di `search-image-sets` codice cerca set di immagini con DICOM Studies generati tra il 1° gennaio 1990 (00:00) e il 1° gennaio 2023 (00:00).

Nota: DICOM è facoltativo. StudyTime Se non è presente, 12:00 AM (inizio della giornata) è il valore temporale per le date fornite per il filtraggio.

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

Contenuto di `search-criteria.json`

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

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

Output:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "updatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

Esempio 3: per cercare set di immagini con un operatore BETWEEN utilizzando CreateDat (gli studi temporali erano precedentemente persistenti)

Il seguente esempio di `search-image-sets` codice cerca set di immagini con DICOM Studies persistenti HealthImaging tra gli intervalli di tempo del fuso orario UTC.

Nota: fornire CreatedAt in un formato di esempio («1985-04-12T 23:20:50.52 Z»).

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

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

Contenuto di `search-criteria.json`

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

Output:

```
{
    "imageSetsMetadataSummaries": [{
        "imageSetId": "09876543210987654321098765432109",
        "createdAt": "2022-12-06T21:40:59.429000+00:00",
        "version": 1,
        "DICOMTags": {
            "DICOMStudyId": "2011201407",
            "DICOMStudyDate": "19991122",
            "DICOMPatientSex": "F",
            "DICOMStudyInstanceUID": "1.2.840.99999999.84710745.943275268089",
            "DICOMPatientBirthDate": "19201120",
            "DICOMStudyDescription": "UNKNOWN",
            "DICOMPatientId": "SUBJECT08701",
            "DICOMPatientName": "Melissa844 Huel628",
            "DICOMNumberOfStudyRelatedInstances": 1,
            "DICOMStudyTime": "140728",
            "DICOMNumberOfStudyRelatedSeries": 1
        },
        "lastUpdatedAt": "2022-12-06T21:40:59.429000+00:00"
    }]
}
```

Esempio 4: cercare set di immagini con un operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordinare la risposta in ordine ASC nel campo updatedAt

Il seguente esempio di `search-image-sets` codice cerca i set di immagini con un operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC sul campo updatedAt.

Nota: fornire UpdatedAt in un formato di esempio («1985-04-12T 23:20:50.52 Z»).

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

Contenuto di `search-criteria.json`

```
{
    "filters": [{
        "values": [{
            "updatedAt": "2024-03-11T15:00:05.074000-07:00"
        }, {
            "updatedAt": "2024-03-11T16:00:05.074000-07:00"
        }],
        "operator": "BETWEEN"
    }, {
        "values": [{
            "DICOMSeriesInstanceUID": "1.2.840.99999999.84710745.943275268089"
        }],
        "operator": "EQUAL"
    }],
    "sort": {
        "sortField": "updatedAt",
        "sortOrder": "ASC"
    }
}
```

Output:

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

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

Per ulteriori informazioni, consulta Searching image sets nella Developer Guide.AWS HealthImaging

- Per i dettagli sull'API, consulta SearchImageSets AWS CLICommand Reference.

Java

SDK per Java 2.x

La funzione di utilità per la ricerca di set di immagini.

```
    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;
    }
```

Caso d'uso #1: operatore EQUAL.

```
        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();
        }
```

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

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

 .dicomStudyDateAndTime(DICOMStudyDateAndTime.builder()
```

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

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

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

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

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano precedentemente persistenti.

```
        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();
        }
```

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
        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();
        }
```

- Per i dettagli sull'API, vedere in API Reference. SearchImageSetsAWS SDK for Java 2.x

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

La funzione di utilità per la ricerca di set di immagini.

```
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;
};
```

Caso d'uso #1: operatore EQUAL.

```
const datastoreId = "12345678901234567890123456789012";

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

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

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

```
const datastoreId = "12345678901234567890123456789012";

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

```
    };

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

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano precedentemente persistenti.

```
const datastoreId = "12345678901234567890123456789012";

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

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

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
const datastoreId = "12345678901234567890123456789012";

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

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

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

- Per i dettagli sull'API, vedere in API Reference. SearchImageSetsAWS SDK for JavaScript

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

La funzione di utilità per la ricerca di set di immagini.

```
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
```

Caso d'uso #1: operatore EQUAL.

```
       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}")
```

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

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

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

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano precedentemente persistenti.

```
        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}"
        )
```

Caso d'uso #4: operatore EQUAL su DICOM SeriesInstance UID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
        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}")
```

Il codice seguente crea un'istanza dell'oggetto. MedicalImagingWrapper

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

- Per i dettagli sull'API, consulta SearchImageSets AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`StartDICOMImportJob`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`StartDICOMImportJob`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. Puoi vedere questa azione nel contesto nel seguente esempio di codice:

- Inizia con set di immagini e cornici di immagini

C++

SDK per C++

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

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

    if (startDICOMImportJobOutcome.IsSuccess()) {
        importJobId = startDICOMImportJobOutcome.GetResult().GetJobId();
    }
    else {
```

```
        std::cerr << "Failed to start DICOM import job because "
                  << startDICOMImportJobOutcome.GetError().GetMessage() <<
 std::endl;
    }

    return startDICOMImportJobOutcome.IsSuccess();
}
```

- Per i dettagli sull'API, consulta StartDicom ImportJob in AWS SDK for C++ API Reference.

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

CLI

AWS CLI

Per avviare un processo di importazione di file dicom

Il seguente esempio di `start-dicom-import-job` codice avvia un processo di importazione dicom.

```
aws medical-imaging start-dicom-import-job \
    --job-name "my-job" \
    --datastore-id "12345678901234567890123456789012" \
    --input-s3-uri "s3://medical-imaging-dicom-input/dicom_input/" \
    --output-s3-uri "s3://medical-imaging-output/job_output/" \
    --data-access-role-arn "arn:aws:iam::123456789012:role/
ImportJobDataAccessRole"
```

Output:

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

```
}
```

Per ulteriori informazioni, consulta Avvio di un processo di importazione nella Guida per gli AWS HealthImaging sviluppatori.

- Per i dettagli sull'API, consulta StartDicom ImportJob in AWS CLI Command Reference.

Java

SDK per Java 2.x

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

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

       return "";
   }
```

- Per i dettagli sull'API, consulta StartDicom ImportJob in API Reference.AWS SDK for Java 2.x

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

```
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- Per i dettagli sull'API, consulta StartDicom ImportJob in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

        :param job_name: The name of the job.
        :param datastore_id: The ID of the data store.
```

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta StartDICOM ImportJob nella guida di riferimento all'API AWS SDK for Python (Boto3).

> **Note**
>
> C' GitHubè di più su. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`TagResource`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`TagResource`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Taggare un archivio dati
- Etichettare un set di immagini

CLI

AWS CLI

Esempio 1: etichettare un archivio dati

I seguenti esempi di `tag-resource` codice contrassegnano un data store.

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

Questo comando non produce alcun output.

Esempio 2: etichettare un set di immagini

I seguenti esempi di `tag-resource` codice contrassegnano un set di immagini.

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

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Tagging resources with AWS HealthImaging nella AWS HealthImaging Developer Guide.

- Per i dettagli sull'API, consulta TagResource AWS CLICommand Reference.

Java

SDK per Java 2.x

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

           medicalImagingClient.tagResource(tagResourceRequest);

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

- Per i dettagli sull'API, TagResourceconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

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

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [TagResource](#)Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta TagResource AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UntagResource`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`UntagResource`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Taggare un archivio dati
- Etichettare un set di immagini

CLI

AWS CLI

Esempio 1: rimuovere i tag da un archivio dati

Il seguente esempio di `untag-resource` codice toglie i tag a un data store.

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

Questo comando non produce alcun output.

Esempio 2: rimuovere i tag da un set di immagini

Il seguente esempio di `untag-resource` codice toglie i tag a un set di immagini.

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

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Tagging resources with AWS HealthImaging nella Developer Guide.AWS HealthImaging

- Per i dettagli sull'API, consulta UntagResource AWS CLICommand Reference.

Java

SDK per Java 2.x

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

           medicalImagingClient.untagResource(untagResourceRequest);

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

- Per i dettagli sull'API, UntagResourceconsulta AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

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

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API UntagResourceReference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

- Per i dettagli sull'API, consulta UntagResource AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateImageSetMetadata`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`UpdateImageSetMetadata`.

CLI

AWS CLI

Per inserire o aggiornare un attributo nei metadati del set di immagini

Il seguente esempio di `update-image-set-metadata` codice inserisce o aggiorna un attributo nei metadati del set di immagini.

```
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
```

Contenuto di `metadata-updates.json`

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

Nota: `updatableAttributes` è una stringa JSON con codifica Base64. Ecco la stringa JSON non codificata.

{» SchemaVersion «:1.1, "Paziente»: {"DICOM»: {» «:"MX^MX"PatientName}}}

Output:

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

Per rimuovere un attributo dai metadati del set di immagini

Il seguente esempio di `update-image-set-metadata` codice rimuove un attributo dai metadati del set di immagini.

```
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
```

Contenuto di `metadata-updates.json`

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

Nota: `removableAttributes` è una stringa JSON con codifica Base64. Ecco la stringa JSON non codificata. La chiave e il valore devono corrispondere all'attributo da rimuovere.

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

Output:

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

Per rimuovere un'istanza dai metadati del set di immagini

Il seguente esempio di `update-image-set-metadata` codice rimuove un'istanza dai metadati del set di immagini.

```
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
```

Contenuto di `metadata-updates.json`

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

Nota: `removableAttributes` è una stringa JSON con codifica Base64. Ecco la stringa JSON non codificata.

{"1.1.1.1.1.12345.123456789012.123.12345678901234.1": {" Istanze»: {"1.1.1.1.1.12345.123456789012.123.1234567890123.1234567890123.12345678901234.1": {}}}}}

Output:

```
{
```

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

Per ulteriori informazioni, consulta Aggiornamento dei metadati del set di immagini nella Guida per gli sviluppatori.AWS HealthImaging

- Per i dettagli sull'API, consulta UpdateImageSetMetadata AWS CLICommand Reference.

Java

SDK per Java 2.x

```
   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);
       }
```

```
    }
```

Caso d'uso #1: Inserimento o aggiornamento di un attributo.

```
            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);
```

Caso d'uso #2: rimuovere un attributo.

```
            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);
```

Caso d'uso #3: rimuove un'istanza.

```
            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);
```

- Per i dettagli sull'API, consulta la UpdateImageSetMetadatasezione AWS SDK for Java 2.x API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

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

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set
 version.
 * @param {{}} updateMetadata - The metadata to update.
 */
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;
};
```

Caso d'uso #1: inserire o aggiornare un attributo.

```
        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);
```

Caso d'uso #2: rimuovere un attributo.

```
        // 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);
```

Caso d'uso #3: rimuove un'istanza.

```
        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);
```

- Per i dettagli sull'API, consulta la UpdateImageSetMetadatasezione AWS SDK for JavaScript API Reference.

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

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


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

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

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

Il codice seguente crea un'istanza dell' MedicalImagingWrapper oggetto.

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

Caso d'uso #1: Inserimento o aggiornamento di un attributo.

```
            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
            )
```

Caso d'uso #2: rimuovere un attributo.

```
            # 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
            )
```

Caso d'uso #3: rimuove un'istanza.

```
            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
            )
```

- Per i dettagli sull'API, consulta UpdateImageSetMetadata AWSSDK for Python (Boto3) API Reference.

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scenari per l' HealthImaging utilizzo AWS degli SDK

I seguenti esempi di codice mostrano come implementare scenari comuni HealthImaging con gli AWS SDK. Questi scenari mostrano come eseguire attività specifiche richiamando più funzioni all'interno. HealthImaging Ogni scenario include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice.

Esempi

- Inizia a usare set di HealthImaging immagini e cornici di immagini utilizzando un AWS SDK
- Taggare un HealthImaging data store utilizzando un SDK AWS
- Taggare un set di HealthImaging immagini utilizzando un SDK AWS

Inizia a usare set di HealthImaging immagini e cornici di immagini utilizzando un AWS SDK

I seguenti esempi di codice mostrano come importare file DICOM e scaricare frame di immagini in. HealthImaging

L'implementazione è strutturata come un'applicazione a riga di comando per il flusso di lavoro.

- Imposta le risorse per un'importazione DICOM.
- Importa file DICOM in un archivio dati.
- Recupera gli ID del set di immagini per il processo di importazione.
- Recupera gli ID dei frame di immagine per i set di immagini.
- Scarica, decodifica e verifica i frame dell'immagine.
- Pulisci le risorse.

C++

SDK per C++

Crea uno AWS CloudFormation stack con le risorse necessarie.

```
   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: ");
    }
```

Copia i file DICOM nel bucket di importazione di Amazon S3.

```
    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;
    }
```

Importa i file DICOM nell'archivio dati Amazon S3.

```
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;
}
```

Ottieni i set di immagini creati dal processo di importazione DICOM.

```
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;
}
```

Ottieni informazioni sulla cornice dell'immagine per i set di immagini.

```
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();
}
```

Scarica, decodifica e verifica i frame delle immagini.

```
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;
    }
```

Pulisci le risorse.

```
    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;
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for C++ .
  - DeleteImageSet
  - Ottieni DICOM ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Avvia DICOM ImportJob

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

index.js- Orchestra i passaggi.

```
// 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- Distribuisci risorse.

```
// 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- Copiare i file DICOM.

```
// 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- Inizia l'importazione nel datastore.

```
// 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- Ottieni gli ID dei set di immagini.

```
// 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- Ottieni gli ID delle cornici delle immagini.

```
// 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- Verifica le cornici delle immagini. La libreria AWS HealthImaging Pixel Data Verification è stata utilizzata per la verifica.

```
// 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- Distruggi le risorse.

```
// 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,
  },
);
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - DeleteImageSet
  - Ottieni DICOM ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Avvia DICOM ImportJob

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

Crea uno AWS CloudFormation stack con le risorse necessarie.

```
    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
```

Copia i file DICOM nel bucket di importazione di Amazon S3.

```
    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.")
```

Importa i file DICOM nell'archivio dati Amazon S3.

```


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"]
```

Ottieni i set di immagini creati dal processo di importazione DICOM.

```
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
```

Ottieni informazioni sulla cornice dell'immagine per i set di immagini.

```
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
```

Scarica, decodifica e verifica i frame delle immagini.

```
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
```

Pulisci le risorse.

```
    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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - DeleteImageSet
  - Ottieni DICOM ImportJob

- GetImageFrame
- GetImageSetMetadata
- SearchImageSets
- Avvia DICOM ImportJob

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Taggare un HealthImaging data store utilizzando un SDK AWS

I seguenti esempi di codice mostrano come etichettare un archivio HealthImaging dati.

Java

SDK per Java 2.x

Per etichettare un archivio dati.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

La funzione di utilità per etichettare una risorsa.

```
   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);
        }
    }
```

Per elencare i tag per un archivio dati.

```
                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());
                }
```

La funzione di utilità per elencare i tag di una risorsa.

```
    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;
    }
```

Per rimuovere i tag da un archivio dati.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

La funzione di utilità per rimuovere il tag di una risorsa.

```
    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);
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

Per etichettare un archivio dati.

```
  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);
  }
```

La funzione di utilità per etichettare una risorsa.

```
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;
};
```

Per elencare i tag per un archivio dati.

```
  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);
  }
```

La funzione di utilità per elencare i tag di una risorsa.

```
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;
};
```

Per rimuovere i tag da un archivio dati.

```
  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);
  }
```

La funzione di utilità per rimuovere il tag di una risorsa.

```
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;
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

Per etichettare un archivio dati.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

La funzione di utilità per etichettare una risorsa.

```
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
```

Per elencare i tag per un archivio dati.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

La funzione di utilità per elencare i tag di una risorsa.

```
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"]
```

Per rimuovere i tag da un archivio dati.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

La funzione di utilità per rimuovere il tag di una risorsa.

```
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
```

Il codice seguente crea un'istanza dell'oggetto. MedicalImagingWrapper

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Taggare un set di HealthImaging immagini utilizzando un SDK AWS

I seguenti esempi di codice mostrano come etichettare un set di HealthImaging immagini.

Java

SDK per Java 2.x

Per etichettare un set di immagini.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

La funzione di utilità per etichettare una risorsa.

```
    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);
        }
    }
```

Per elencare i tag per un set di immagini.

```
                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());
                }
```

La funzione di utilità per elencare i tag di una risorsa.

```
    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;
    }
```

Per rimuovere i tag da un set di immagini.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                Collections.singletonList("Deployment"));
```

La funzione di utilità per rimuovere il tag di una risorsa.

```
    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);
        }
```

```
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

JavaScript

SDK per JavaScript (v3)

Per etichettare un set di immagini.

```
  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);
  }
```

La funzione di utilità per etichettare una risorsa.

```
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;
};
```

Per elencare i tag per un set di immagini.

```
  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);
  }
```

La funzione di utilità per elencare i tag di una risorsa.

```
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;
};
```

Per rimuovere i tag da un set di immagini.

```
  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);
  }
```

La funzione di utilità per rimuovere il tag di una risorsa.

```
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;
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource

- UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Python

SDK per Python (Boto3)

Per etichettare un set di immagini.

```
    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"})
```

La funzione di utilità per etichettare una risorsa.

```
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
```

Per elencare i tag per un set di immagini.

```
    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)
```

La funzione di utilità per elencare i tag di una risorsa.

```
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"]
```

Per rimuovere i tag da un set di immagini.

```
    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"])
```

La funzione di utilità per rimuovere il tag di una risorsa.

```
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
```

Il codice seguente crea un'istanza dell'oggetto. MedicalImagingWrapper

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo HealthImaging con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Monitoraggio di AWS HealthImaging

Il monitoraggio e la registrazione sono elementi importanti per mantenere la sicurezza, l'affidabilità, la disponibilità e le prestazioni di AWS HealthImaging. AWS fornisce i seguenti strumenti di registrazione e monitoraggio per osservare HealthImaging, segnalare quando qualcosa non va e intraprendere azioni automatiche se necessario:

- AWS CloudTrailacquisisce le chiamate API e gli eventi correlati effettuati da o per conto del tuo AWS account e invia i file di log a un bucket Amazon S3 da te specificato. Puoi identificare quali utenti e account hanno chiamato AWS, l'indirizzo IP di origine da cui sono state effettuate le chiamate e quando sono avvenute le chiamate. Per ulteriori informazioni, consulta la Guida per l'utente AWS CloudTrail.
- Amazon CloudWatch monitora AWS le tue risorse e le applicazioni su cui esegui AWS in tempo reale. Puoi raccogliere i parametri e tenerne traccia, creare pannelli di controllo personalizzati e impostare allarmi per inviare una notifica o intraprendere azioni quando un parametro specificato raggiunge una determinata soglia. Ad esempio, puoi tenere CloudWatch traccia dell'utilizzo della CPU o di altri parametri delle tue istanze Amazon EC2 e avviare automaticamente nuove istanze quando necessario. Per ulteriori informazioni, consulta la Amazon CloudWatch User Guide.
- Amazon EventBridge è un servizio di bus eventi senza server che semplifica la connessione delle applicazioni con dati provenienti da una varietà di fonti. EventBridge fornisce un flusso di dati in tempo reale dalle tue applicazioni, dalle applicazioni oftware-as-a S-Service (SaaS) e dai servizi AWS e indirizza tali dati verso destinazioni come Lambda. In questo modo puoi monitorare gli eventi che si verificano nei servizi e creare architetture basate su eventi. Per ulteriori informazioni, consulta la Amazon EventBridge User Guide.

Argomenti

Utilizzo AWS CloudTrail con HealthImaging

AWS HealthImaging è integrato con AWS CloudTrail, un servizio che fornisce un registro delle azioni intraprese da un utente, un ruolo o un AWS servizio in HealthImaging. CloudTrail acquisisce

tutte le chiamate API HealthImaging come eventi. Le chiamate acquisite includono chiamate dalla HealthImaging console e chiamate di codice alle operazioni HealthImaging API. Se crei un trail, puoi attivare la distribuzione continua di CloudTrail eventi a un bucket Amazon S3, inclusi gli eventi per. HealthImaging Se non configuri un percorso, puoi comunque visualizzare gli eventi più recenti nella CloudTrail console nella cronologia degli eventi. Utilizzando le informazioni raccolte da CloudTrail, puoi determinare a quale richiesta è stata inviata HealthImaging, l'indirizzo IP da cui è stata effettuata la richiesta, chi ha effettuato la richiesta, quando è stata effettuata e dettagli aggiuntivi.

Per ulteriori informazioni CloudTrail, consulta la Guida AWS CloudTrail per l'utente.

Creazione di un percorso

CloudTrail viene attivata per te Account AWS quando crei l'account. Quando si verifica un'attività in HealthImaging, tale attività viene registrata in un CloudTrail evento insieme ad altri eventi AWS di servizio nella cronologia degli eventi. Puoi visualizzare, cercare e scaricare eventi recenti in Account AWS. Per ulteriori informazioni, consulta Visualizzazione degli eventi con la cronologia degli CloudTrail eventi.

> **Note**
>
> Per visualizzare la cronologia degli CloudTrail eventi per AWS HealthImaging in AWS Management Console, vai al menu Lookup attributes, seleziona Event source e scegli`medical-imaging.amazonaws.com`.

Per una registrazione continua degli eventi del tuo sito Account AWS, inclusi gli eventi di HealthImaging, crea un percorso. Un trail consente di CloudTrail inviare file di log a un bucket Amazon S3. Per impostazione predefinita, quando si crea un percorso nella console, questo sarà valido in tutte le Regioni AWS. Il trail registra gli eventi di tutte le regioni della AWS partizione e consegna i file di log al bucket Amazon S3 specificato. Inoltre, puoi configurare altri AWS servizi per analizzare ulteriormente e agire in base ai dati sugli eventi raccolti nei log. CloudTrail Per ulteriori informazioni, consulta gli argomenti seguenti:

- Panoramica della creazione di un percorso
- CloudTrail servizi e integrazioni supportati
- Configurazione delle notifiche Amazon SNS per CloudTrail
- Ricezione di file di CloudTrail registro da più regioni e ricezione di file di CloudTrail registro da più account

> **Note**
>
> AWS HealthImaging supporta due tipi di CloudTrail eventi: eventi di gestione ed eventi relativi ai dati. Gli eventi di gestione sono gli eventi generali generati da ogni AWS servizio, inclusi HealthImaging. Per impostazione predefinita, la registrazione viene applicata agli eventi di gestione per ogni chiamata HealthImaging API in cui è abilitata. Gli eventi relativi ai dati sono fatturabili e generalmente riservati alle API con transazioni al secondo (tps) elevate, quindi puoi scegliere di non avere CloudTrail registri a fini di costo.
> Con HealthImaging, tutte le azioni API elencate nell'AWS HealthImaging API Reference sono considerate eventi di gestione ad eccezione di `GetImageFrame`. L'`GetImageFrame`azione viene inserita CloudTrail come evento di dati e pertanto deve essere abilitata. Per ulteriori informazioni, consultare Registrazione di eventi di dati nella Guida per l'utente di AWS CloudTrail .

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con credenziali utente root o AWS Identity and Access Management (IAM).
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro AWS servizio.

Per ulteriori informazioni, vedete l'CloudTrail `userIdentity`elemento.

Comprensione delle voci di registro

Un trail è una configurazione che consente la distribuzione di eventi come file di log in un bucket Amazon S3 specificato dall'utente. CloudTrail i file di registro contengono una o più voci di registro. Un evento rappresenta una singola richiesta proveniente da qualsiasi fonte e include informazioni sull'azione richiesta, la data e l'ora dell'azione, i parametri della richiesta e così via. CloudTrail i file di registro non sono una traccia ordinata dello stack delle chiamate API pubbliche, quindi non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra una voce di CloudTrail registro HealthImaging che illustra l'`GetDICOMImportJob`azione.

```
{
    "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"
}
```

Usare Amazon CloudWatch con HealthImaging

Puoi monitorare AWS HealthImaging utilizzando CloudWatch, che raccoglie dati grezzi e li elabora in metriche leggibili quasi in tempo reale. Queste statistiche vengono conservate per 15 mesi, quindi puoi utilizzare tali informazioni storiche e avere una prospettiva migliore sulle prestazioni della tua applicazione o del tuo servizio web. È anche possibile impostare allarmi che controllano determinate soglie e inviare notifiche o intraprendere azioni quando queste soglie vengono raggiunte. Per ulteriori informazioni, consulta la Amazon CloudWatch User Guide.

> **Note**
>
> Le metriche vengono riportate per tutte le HealthImaging API.

Le tabelle seguenti elencano le metriche e le dimensioni per. HealthImaging Ciascuna viene presentata come conteggio delle frequenze per un intervallo di dati specificato dall'utente.

Metriche

Metriche	Descrizione
Numero di chiamate	Il numero di chiamate alle API. Questo può essere segnalato per l'account o per un archivio dati specificato. Unità: numero Statistiche valide: somma, conteggio Dimensioni: funzionamento, ID dell'archivio dati, tipo di archivio dati

Puoi ottenere metriche per HealthImaging AWS Management Console AWS CLI, the o l' CloudWatch API. Puoi utilizzare l' CloudWatch API tramite uno degli Amazon AWS Software Development Kit (SDK) o gli strumenti CloudWatch API. La HealthImaging console mostra grafici basati sui dati grezzi dell'API. CloudWatch

È necessario disporre delle CloudWatch autorizzazioni appropriate con cui eseguire il monitoraggio HealthImaging . CloudWatch Per ulteriori informazioni, consulta la sezione Gestione delle identità e degli accessi CloudWatch nella Guida per l'CloudWatch utente.

Visualizzazione delle HealthImaging metriche

Per visualizzare le metriche (console) CloudWatch

1. Accedi a AWS Management Console e apri la CloudWatch console.
2. Scegli Metriche, scegli Tutte le metriche, quindi scegli AWS/Medical Imaging.
3. Scegliere la dimensione, selezionare un nome parametro e scegliere Add to graph (Aggiungi a grafico).
4. Seleziona un valore per l'intervallo di date. Il numero di parametri per l'intervallo di date selezionato è visualizzato nel grafico.

Creazione di un allarme utilizzando CloudWatch

Un CloudWatch allarme controlla una singola metrica in un periodo di tempo specificato ed esegue una o più azioni: inviare una notifica a un argomento di Amazon Simple Notification Service (Amazon SNS) o a una politica di Auto Scaling. L'azione o le azioni si basano sul valore della metrica relativo a una determinata soglia in un certo numero di periodi di tempo specificati. CloudWatch può anche inviarti un messaggio Amazon SNS quando l'allarme cambia stato.

CloudWatch gli allarmi richiamano azioni solo quando lo stato cambia e persiste per il periodo specificato. Per ulteriori informazioni, consulta Uso degli allarmi. CloudWatch

Usare Amazon EventBridge con HealthImaging

Amazon EventBridge è un servizio serverless che utilizza gli eventi per connettere tra loro i componenti delle applicazioni, semplificando la creazione di applicazioni scalabili basate sugli eventi. La base di EventBridge è creare regole che indirizzino gli eventi verso gli obiettivi. AWS HealthImaging fornisce una distribuzione duratura delle modifiche di stato a EventBridge. Per ulteriori informazioni, consulta What is Amazon EventBridge? nella Amazon EventBridge User Guide.

Argomenti

- HealthImaging eventi inviati a EventBridge
- HealthImaging struttura degli eventi ed esempi

HealthImaging eventi inviati a EventBridge

La tabella seguente elenca tutti HealthImaging gli eventi inviati EventBridge per l'elaborazione.

HealthImaging tipo di evento	Stato
eventi dell'archivio dati	
Creazione di archivi dati	`CREATING`
Creazione del Data Store non riuscita	`CREATE_FAILED`
Data Store creato	`ACTIVE`
Eliminazione dell'archivio dati	`DELETING`
Archivio dati eliminato	`DELETED`
Per ulteriori informazioni, consulta DataStoreStatus nell'AWS API Reference. HealthImaging	
Importa eventi di lavoro	
Importa Job inviato	`SUBMITTED`
Importazione Job In Progress	`IN_PROGRESS`
Importazione Job completata	`COMPLETED`
Importazione Job non riuscita	`FAILED`
Per ulteriori informazioni, consulta JobStatus nell'AWS HealthImaging API Reference.	
Eventi del set di immagini	
Set di immagini creato	`CREATED`
Copia del set di immagini	`COPYING`
Copia di set di immagini con accesso in sola lettura	`COPYING_WITH_READ_ONLY_ACCESS`
Set di immagini copiato	`COPIED`

HealthImaging tipo di evento	Stato
Copia del set di immagini non riuscita	`COPY_FAILED`
Aggiornamento del set di immagini	`UPDATING`
Set di immagini aggiornato	`UPDATED`
Aggiornamento del set di immagini non riuscito	`UPDATE_FAILED`
Eliminazione del set di immagini	`DELETING`
Set di immagini eliminato	`DELETED`

Per ulteriori informazioni, consulta ImageSetWorkflowStatusl'AWS HealthImaging API Reference.

HealthImaging struttura degli eventi ed esempi

HealthImaging gli eventi sono oggetti con struttura JSON che contengono anche dettagli sui metadati. È possibile utilizzare i metadati come input per ricreare un evento o ottenere ulteriori informazioni. Tutti i campi di metadati associati sono elencati in una tabella sotto gli esempi di codice nei seguenti menu. Per ulteriori informazioni, consulta il riferimento alla struttura degli eventi nella Amazon EventBridge User Guide.

> **Note**
>
> L'`source`attributo per le strutture HealthImaging degli eventi è`aws.medical-imaging`.

Eventi dell'archivio dati

Data Store Creating

Stato - **`CREATING`**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
```

```
    "detail-type": "Data Store Creating",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "CREATING"
    }
}
```

Data Store Creation Failed

Stato - **CREATE_FAILED**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Creation Failed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "CREATE_FAILED"
    }
}
```

Data Store Created

Stato - **ACTIVE**

```
{
    "version": "0",
```

```
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Created",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "ACTIVE"
    }
}
```

Data Store Deleting

Stato - **DELETING**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Deleting",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "DELETING"
    }
}
```

Data Store Deleted

Stato - **DELETED**

```
{
```

```
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Data Store Deleted",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "datastoreName": "test",
        "datastoreStatus": "DELETED"
    }
}
```

Eventi dell'archivio dati: descrizioni dei metadati

Nome	Type	Descrizione
`version`	stringa	La versione dello schema degli EventBridge eventi.
`id`	string	L'UUID della versione 4 generato per ogni evento.
`detail-type`	string	Il tipo di evento che viene inviato.
`source`	string	Identifica il servizio che ha generato l'evento.
`account`	string	L'ID dell'account AWS a 12 cifre del proprietario del data store.
`time`	string	L'ora in cui si è verificato l'evento.

Nome	Type	Descrizione
`region`	string	Identifica la AWS regione dell'archivio dati.
`resources`	array (stringa)	Un array JSON che contiene l'ARN dell'archivio dati.
`detail`	oggetto	Un oggetto JSON contenente informazioni sull'evento.
`detail.imagingVersion`	string	L'ID della versione che tiene traccia delle modifiche allo schema HealthImaging di dettaglio degli eventi.
`detail.datastoreId`	string	L'ID del data store associato all'evento di modifica dello stato.
`detail.datastoreName`	string	Il nome del data store.
`detail.datastoreStatus`	string	Lo stato corrente del data store.

Importa eventi di lavoro

Import Job Submitted

Stato - **`SUBMITTED`**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Submitted",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
```

```
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "SUBMITTED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

Import Job In Progress

Stato - **IN_PROGRESS**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job In Progress",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "IN_PROGRESS",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

Import Job Completed

Stato - **COMPLETED**

```
{
```

```
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Completed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "COMPLETED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
}
```

Import Job Failed

Stato - **`FAILED`**

```
{
    "version": "0",
    "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
    "detail-type": "Import Job Failed",
    "source": "aws.medical-imaging",
    "account": "111122223333",
    "time": "2024-03-14T00:01:00Z",
    "region": "us-west-2",
    "resources": ["arn:aws:medical-imaging:us-west-2:111122223333:datastore/
bbc4f3cccbae4095a34170fddc19b13d"],
    "detail": {
        "imagingVersion": "1.0",
        "datastoreId" : "bbc4f3cccbae4095a34170fddc19b13d",
        "jobId": "a6a1d220f152e7aab6d8925d995d8b76",
        "jobName": "test_only_1",
        "jobStatus": "FAILED",
        "inputS3Uri": "s3://healthimaging-test-bucket/input/",
        "outputS3Uri": "s3://healthimaging-test-bucket/output/"
    }
```

```
}
```

Importa eventi di lavoro - descrizioni dei metadati

Nome	Type	Descrizione
`version`	stringa	La versione dello schema degli EventBridge eventi.
`id`	string	L'UUID della versione 4 generato per ogni evento.
`detail-type`	string	Il tipo di evento che viene inviato.
`source`	string	Identifica il servizio che ha generato l'evento.
`account`	string	L'ID dell'account AWS a 12 cifre del proprietario del data store.
`time`	string	L'ora in cui si è verificato l'evento.
`region`	string	Identifica la AWS regione dell'archivio dati.
`resources`	array (stringa)	Un array JSON che contiene l'ARN dell'archivio dati.
`detail`	oggetto	Un oggetto JSON contenente informazioni sull'evento.
`detail.imagingVersion`	string	L'ID della versione che tiene traccia delle modifiche allo schema HealthImaging di dettaglio degli eventi.

Nome	Type	Descrizione
`detail.datastoreId`	string	L'archivio dati che ha generato l'evento di modifica dello stato.
`detail.jobId`	string	L'ID del processo di importazione associato all'evento di modifica dello stato.
`detail.jobName`	string	Il nome del processo di importazione.
`detail.jobStatus`	string	Lo stato attuale del lavoro.
`detail.inputS3Uri`	string	Il percorso del prefisso di input per il bucket S3 che contiene i file DICOM da importare.
`detail.outputS3Uri`	string	Il prefisso di output del bucket S3 in cui verranno caricati i risultati del processo di importazione DICOM.

Eventi del set di immagini

Image Set Created

Stato - **`CREATED`**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Created",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
```

```
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "CREATED"
   }
}
```

Image Set Copying

Stato - **COPYING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING"
   }
}
```

Image Set Copying With Read Only Access

Stato - **COPYING_WITH_READ_ONLY_ACCESS**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copying With Read Only Access",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
```

```
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "COPYING_WITH_READ_ONLY_ACCESS"
   }
}
```

Image Set Copied

Stato - **COPIED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copied",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPIED"
   }
}
```

Image Set Copy Failed

Stato - **COPY_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Copy Failed",
   "source": "aws.medical-imaging",
```

```
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "COPY_FAILED"
   }
}
```

Image Set Updating

Stato - **UPDATING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updating",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "UPDATING"
   }
}
```

Image Set Updated

Stato - **UPDATED**

```
{
   "version": "0",
```

```
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Updated",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATED"
   }
}
```

Image Set Update Failed

Stato - **UPDATE_FAILED**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Update Failed",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "ACTIVE",
      "imageSetWorkflowStatus": "UPDATE_FAILED"
   }
}
```

Image Set Deleting

Stato - **DELETING**

```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleting",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "LOCKED",
      "imageSetWorkflowStatus": "DELETING"
   }
}
```


Image Set Deleted

Stato - **DELETED**


```
{
   "version": "0",
   "id": "7cf0fb1c-8720-4d48-baa3-6eb97b35a10e",
   "detail-type": "Image Set Deleted",
   "source": "aws.medical-imaging",
   "account": "111122223333",
   "time": "2024-03-14T00:01:00Z",
   "region": "us-west-2",
   "resources": ["arn:aws:medical-imaging:us-west-2:846006145877:datastore/
bbc4f3cccbae4095a34170fddc19b13d/imageset/207284eef860ac01c4b2a8de27a6fc11"],
   "detail": {
      "imagingVersion": "1.0",
      "datastoreId": "bbc4f3cccbae4095a34170fddc19b13d",
      "imagesetId": "5b3a711878c34d40e888253319388649",
      "imageSetState": "DELETED",
      "imageSetWorkflowStatus": "DELETED"
   }
}
```

Eventi dei set di immagini - descrizioni dei metadati

Nome	Type	Descrizione
`version`	stringa	La versione dello schema degli EventBridge eventi.
`id`	string	L'UUID della versione 4 generato per ogni evento.
`detail-type`	string	Il tipo di evento che viene inviato.
`source`	string	Identifica il servizio che ha generato l'evento.
`account`	string	L'ID dell'account AWS a 12 cifre del proprietario del data store.
`time`	string	L'ora in cui si è verificato l'evento.
`region`	string	Identifica la AWS regione dell'archivio dati.
`resources`	array (stringa)	Un array JSON che contiene l'ARN del set di immagini.
`detail`	oggetto	Un oggetto JSON contenente informazioni sull'evento.
`detail.imagingVersion`	string	L'ID della versione che tiene traccia delle modifiche allo schema HealthImaging di dettaglio degli eventi.
`detail.datastoreId`	string	L'ID del data store che ha generato l'evento di modifica dello stato.

Nome	Type	Descrizione
`detail.imagesetId`	string	L'ID del set di immagini associato all'evento di modifica dello stato.
`detail.imageSetState`	string	Lo stato corrente del set di immagini.
`detail.imageSetWorkflowStatus`	string	Lo stato corrente del flusso di lavoro del set di immagini.

Sicurezza in AWS HealthImaging

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di data center e architetture di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra te e te. AWS Il modello di responsabilità condivisa descrive questo aspetto come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi in Cloud AWS. AWS fornisce inoltre servizi che è possibile utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei AWS Programmi di AWS conformità dei Programmi di conformità dei di . Per ulteriori informazioni sui programmi di conformità applicabili AWS HealthImaging, consulta AWS Servizi nell'ambito del programma di conformitàAWS .
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che utilizzi. Sei anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della tua azienda e le leggi e normative vigenti.

Questa documentazione ti aiuta a capire come applicare il modello di responsabilità condivisa durante l'utilizzo HealthImaging. I seguenti argomenti mostrano come eseguire la configurazione HealthImaging per soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche a utilizzare altri AWS servizi che ti aiutano a monitorare e proteggere HealthImaging le tue risorse.

Argomenti

Protezione dei dati in AWS HealthImaging

Il modello di responsabilità AWS condivisa modello di si applica alla protezione dei dati in AWS HealthImaging. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutto il Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per ulteriori informazioni sulla privacy dei dati, vedi le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al Modello di responsabilità condivisa AWS e GDPR nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-2 per l'accesso AWS tramite un'interfaccia a riga di comando o un'API, utilizza un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-2.

Ti consigliamo vivamente di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori o Servizi AWS utilizzi la console, l'API HealthImaging o gli SDK. AWS CLI AWS I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Argomenti

Crittografia dei dati

Con AWS HealthImaging, puoi aggiungere un livello di sicurezza ai tuoi dati archiviati nel cloud, fornendo funzionalità di crittografia scalabili ed efficienti. Ciò include:

- Funzionalità di crittografia dei dati archiviati disponibili nella maggior parte dei servizi AWS
- Opzioni flessibili di gestione delle chiavi AWS Key Management Service, tra cui è possibile scegliere se AWS gestire le chiavi di crittografia o mantenere il controllo completo sulle proprie chiavi.
- AWS chiavi di AWS KMS crittografia possedute
- Code di messaggi crittografate per la trasmissione di dati sensibili utilizzando la crittografia lato server (SSE) per Amazon SQS

Inoltre, AWS fornisce API per integrare la crittografia e la protezione dei dati con qualsiasi servizio sviluppato o distribuito in un ambiente. AWS

Crittografia a riposo

HealthImaging fornisce la crittografia di default per proteggere i dati sensibili dei clienti archiviati utilizzando una chiave di proprietà del servizio AWS KMS .

Crittografia in transito

HealthImaging utilizza TLS 1.2 per crittografare i dati in transito attraverso l'endpoint pubblico e tramite i servizi di backend.

Gestione delle chiavi

AWS KMS le chiavi (chiavi KMS) sono la risorsa principale in AWS Key Management Service. È inoltre possibile generare chiavi di dati da utilizzare all'esterno di AWS KMS.

AWS chiave KMS proprietaria

HealthImaging utilizza queste chiavi per impostazione predefinita per crittografare automaticamente le informazioni potenzialmente sensibili come i dati personali identificabili o i dati PHI (Private Health Information) inattivi. AWS le chiavi KMS di tua proprietà non sono archiviate nel tuo account.

Fanno parte di una raccolta di chiavi KMS che AWS possiede e gestisce per l'utilizzo in più AWS account. AWS i servizi possono utilizzare chiavi KMS di AWS proprietà per proteggere i dati. Non puoi visualizzare, gestire, utilizzare chiavi KMS AWS di proprietà o controllarne l'utilizzo. Tuttavia, non è necessario eseguire alcuna operazione o modificare alcun programma per proteggere le chiavi che crittografano i dati.

Non ti viene addebitato un canone mensile o un canone di utilizzo se utilizzi chiavi KMS di tua AWS proprietà e non vengono conteggiate nelle AWS KMS quote del tuo account. Per ulteriori informazioni, consulta le chiavi di proprietà di AWS nella AWS Key Management Service Developer Guide.

Chiavi KMS gestite dal cliente

HealthImaging supporta l'uso di una chiave KMS simmetrica gestita dal cliente che puoi creare, possedere e gestire per aggiungere un secondo livello di crittografia rispetto alla crittografia di proprietà esistente. AWS Avendo il pieno controllo di questo livello di crittografia, è possibile eseguire operazioni quali:

- Stabilire e mantenere politiche chiave, politiche IAM e sovvenzioni
- Ruotare i materiali crittografici delle chiavi
- Abilitare e disabilitare le policy delle chiavi
- Aggiungere tag
- Creare alias delle chiavi
- Pianificare l’eliminazione delle chiavi

Puoi anche utilizzarlo CloudTrail per tenere traccia delle richieste HealthImaging inviate a per tuo AWS KMS conto. AWS KMS Si applicano costi aggiuntivi. Per ulteriori informazioni, consulta Customer managed keys nella Guida per sviluppatori AWS Key Management Service .

Creazione di una chiave gestita dal cliente

È possibile creare una chiave simmetrica gestita dal cliente utilizzando le AWS Management Console o le AWS KMS API. Per ulteriori informazioni, consulta Creazione di chiavi KMS di crittografia simmetrica nella Guida per gli sviluppatori.AWS Key Management Service

Le policy della chiave controllano l'accesso alla chiave gestita dal cliente. Ogni chiave gestita dal cliente deve avere esattamente una policy della chiave, che contiene istruzioni che determinano chi può usare la chiave e come la possono usare. Quando crei la chiave gestita dal cliente, puoi

specificare una policy della chiave. Per ulteriori informazioni, consulta Gestione dell'accesso alle chiavi gestite dal cliente nella Guida per gli sviluppatori di AWS Key Management Service .

Per utilizzare la chiave gestita dal cliente con HealthImaging le tue risorse, kms: CreateGrant le operazioni devono essere consentite nella policy chiave. Ciò aggiunge una concessione a una chiave gestita dal cliente che controlla l'accesso a una chiave KMS specificata, che fornisce all'utente l'accesso alle operazioni HealthImaging Grant richieste. Per ulteriori informazioni, consulta Grants AWS KMS nella AWS Key Management Service Developer Guide.

Per utilizzare la chiave KMS gestita dal cliente con HealthImaging le tue risorse, nella policy chiave devono essere consentite le seguenti operazioni API:

- `kms:DescribeKey`fornisce i dettagli chiave gestiti dal cliente necessari per convalidare la chiave. Ciò è necessario per tutte le operazioni.
- `kms:GenerateDataKey`fornisce l'accesso alle risorse di crittografia a riposo per tutte le operazioni di scrittura.
- `kms:Decrypt`fornisce l'accesso alle operazioni di lettura o ricerca per risorse crittografate.
- `kms:ReEncrypt*`fornisce l'accesso alle risorse di ricrittografia.

Di seguito è riportato un esempio di dichiarazione politica che consente a un utente di creare e interagire con un archivio dati in HealthImaging cui è crittografato mediante tale chiave:

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

Autorizzazioni IAM richieste per l'utilizzo di una chiave KMS gestita dal cliente

Quando si crea un data store con AWS KMS crittografia abilitata utilizzando una chiave KMS gestita dal cliente, sono necessarie le autorizzazioni sia per la policy chiave che per la policy IAM per l'utente o il ruolo che crea il data store. HealthImaging

Per ulteriori informazioni sulle politiche chiave, consulta Enabling IAM policies nella AWS Key Management Service Developer Guide.

L'utente IAM, il ruolo IAM o l' AWS account che crea i tuoi repository deve disporre delle autorizzazioni per`kms:CreateGrant`,`kms:GenerateDataKey`,`kms:RetireGrant`, e `kms:Decryptkms:ReEncrypt*`, oltre alle autorizzazioni necessarie per AWS. HealthImaging

Come utilizza le sovvenzioni HealthImaging in AWS KMS

HealthImaging richiede una concessione per utilizzare la chiave KMS gestita dal cliente. Quando crei un archivio dati crittografato con una chiave KMS gestita dal cliente, HealthImaging crea una concessione per tuo conto inviando una CreateGrantrichiesta a. AWS KMS Le sovvenzioni AWS KMS vengono utilizzate per HealthImaging consentire l'accesso a una chiave KMS in un account cliente.

Le sovvenzioni HealthImaging create per tuo conto non devono essere revocate o ritirate. Se revochi o ritiri la concessione che HealthImaging autorizza l'uso delle AWS KMS chiavi del tuo account, HealthImaging non puoi accedere a questi dati, crittografare le nuove risorse di imaging trasferite nell'archivio dati o decrittografarle quando vengono estratte. Quando si revoca o si ritira una sovvenzione, la modifica avviene immediatamente. HealthImaging Per revocare i diritti di accesso, è necessario eliminare l'archivio dati anziché revocare la concessione. Quando un data store viene eliminato, annulla le HealthImaging concessioni per tuo conto.

Monitoraggio delle chiavi di crittografia per HealthImaging

Puoi utilizzarlo CloudTrail per tenere traccia delle richieste HealthImaging inviate a per tuo AWS KMS conto quando utilizzi una chiave KMS gestita dal cliente. Le voci di registro nel CloudTrail registro vengono visualizzate `medical-imaging.amazonaws.com` nel `userAgent` campo per distinguere chiaramente le richieste effettuate da HealthImaging.

Gli esempi seguenti sono CloudTrail eventi per`CreateGrant`, `GenerateDataKeyDecrypt`, e per `DescribeKey` monitorare AWS KMS le operazioni richieste HealthImaging per accedere ai dati crittografati dalla chiave gestita dal cliente.

Di seguito viene illustrato come utilizzare `CreateGrant` per consentire l'accesso HealthImaging a una chiave KMS fornita dal cliente, che consente di HealthImaging utilizzare tale chiave KMS per crittografare tutti i dati inattivi del cliente.

Gli utenti non sono tenuti a creare le proprie sovvenzioni. HealthImaging crea una sovvenzione per tuo conto inviando una `CreateGrant` richiesta a AWS KMS. Le sovvenzioni AWS KMS vengono utilizzate per HealthImaging consentire l'accesso a una AWS KMS chiave in un account cliente.

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

Gli esempi seguenti mostrano come `GenerateDataKey` garantire che l'utente disponga delle autorizzazioni necessarie per crittografare i dati prima di archiviarli.

```
{
    "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"
}
```

L'esempio seguente mostra come HealthImaging richiama l'`Decrypt`operazione per utilizzare la chiave di dati crittografati archiviata per accedere ai dati crittografati.

```
{
    "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"
}
```

L'esempio seguente mostra come HealthImaging utilizza l'`DescribeKey`operazione per verificare se la AWS KMS chiave di proprietà AWS KMS del cliente è in uno stato utilizzabile e per aiutare l'utente a risolvere i problemi se non funziona.

```
{
    "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"
}
```

Ulteriori informazioni

Le seguenti risorse forniscono ulteriori informazioni sulla crittografia dei dati a riposo e sono disponibili nella Guida per gli AWS Key Management Service sviluppatori.

- AWS KMS concetti
- Le migliori pratiche di sicurezza per AWS KMS

Privacy del traffico di rete

Il traffico è protetto sia tra HealthImaging le applicazioni locali che tra HealthImaging Amazon S3. Il traffico tra HealthImaging e AWS Key Management Service utilizza HTTPS per impostazione predefinita.

- AWS HealthImaging è un servizio regionale disponibile nelle regioni Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon), Europa (Irlanda) e Asia Pacifico (Sydney).
- Per il traffico tra HealthImaging e i bucket Amazon S3, Transport Layer Security (TLS) crittografa gli oggetti in transito tra HealthImaging Amazon S3 e tra le applicazioni dei clienti che vi accedono, dovresti consentire solo connessioni crittografate su HTTPS (TLS) utilizzando le policy IAM del `aws:SecureTransport condition`bucket Amazon S3. HealthImaging Sebbene HealthImaging attualmente utilizzi l'endpoint pubblico per accedere ai dati nei bucket Amazon S3, ciò non significa che i dati attraversino la rete Internet pubblica. Tutto il traffico tra Amazon S3 HealthImaging e Amazon S3 viene instradato sulla AWS rete e crittografato tramite TLS.

Identity and Access Management per AWS HealthImaging

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) a utilizzare le risorse. HealthImaging IAM è uno Servizio AWS strumento che puoi utilizzare senza costi aggiuntivi.

Argomenti

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro che HealthImaging svolgi.

Utente del servizio: se utilizzi il HealthImaging servizio per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più HealthImaging funzionalità per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni corrette all'amministratore. Se non riesci ad accedere a una funzionalità in HealthImaging, consultaRisoluzione dei problemi di HealthImaging identità e accesso AWS.

Amministratore del servizio: se sei responsabile delle HealthImaging risorse della tua azienda, probabilmente hai pieno accesso a HealthImaging. È tuo compito determinare a quali HealthImaging funzionalità e risorse devono accedere gli utenti del servizio. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con HealthImaging, consultaIn che modo AWS HealthImaging funziona con IAM.

Amministratore IAM: se sei un amministratore IAM, potresti voler conoscere i dettagli su come scrivere policy a cui gestire l'accesso HealthImaging. Per visualizzare esempi di policy HealthImaging basate sull'identità che puoi utilizzare in IAM, consulta. Esempi di policy basate sull'identità per AWS HealthImaging

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente.

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sull'utilizzo del metodo consigliato per firmare autonomamente le richieste, consulta Signing AWS API request nella IAM User Guide.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta Autenticazione a più fattori nella Guida per l'utente di AWS IAM Identity Center e Utilizzo dell'autenticazione a più fattori (MFA) in AWSnella Guida per l'utente IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzale per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come procedura consigliata, richiedi agli utenti umani, compresi gli utenti che richiedono l'accesso come amministratore, di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente dell'elenco utenti aziendale, di un provider di identità Web AWS Directory Service, della directory Identity Center o di qualsiasi utente che accede utilizzando le Servizi AWS credenziali fornite tramite un'origine di identità. Quando le identità federate accedono Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Puoi creare utenti e gruppi in IAM Identity Center oppure puoi connetterti e sincronizzarti con un set di utenti e gruppi nella tua fonte di identità per utilizzarli su tutte le tue applicazioni. Account AWS Per ulteriori informazioni su IAM Identity Center, consulta Cos'è IAM Identity Center? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, se si hanno casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente IAM.

Un gruppo IAM è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, è possibile avere un gruppo denominato IAMAdmins e concedere a tale gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta Quando creare un utente IAM (invece di un ruolo) nella Guida per l'utente IAM.

Ruoli IAM

Un ruolo IAM è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Puoi assumere temporaneamente un ruolo IAM in AWS Management Console cambiando ruolo. Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- Accesso utente federato: per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta Creazione di un ruolo per un provider di identità di terza parte nella Guida per l'utente IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
- Autorizzazioni utente IAM temporanee: un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso multi-account: è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per conoscere la differenza tra ruoli e politiche basate sulle risorse per l'accesso tra account diversi, consulta Cross Account Resource Access in IAM nella IAM User Guide.
- Accesso tra servizi: alcuni Servizi AWS utilizzano funzionalità in altri. Servizi AWS Ad esempio, quando effettui una chiamata in un servizio, è comune che tale servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
  - Sessioni di accesso diretto (FAS): quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un preside. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama an Servizio AWS, combinate con la richiesta Servizio AWS per

effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina Forward access sessions.

- Ruolo di servizio: un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire azioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWSnella Guida per l'utente IAM.
- Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.

- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un'istanza EC2 e che AWS CLI effettuano richieste API. AWS Ciò è preferibile all'archiviazione delle chiavi di accesso nell'istanza EC2. Per assegnare un AWS ruolo a un'istanza EC2 e renderlo disponibile per tutte le sue applicazioni, crei un profilo di istanza collegato all'istanza. Un profilo dell'istanza contiene il ruolo e consente ai programmi in esecuzione sull'istanza EC2 di ottenere le credenziali temporanee. Per ulteriori informazioni, consulta Utilizzo di un ruolo IAM per concedere autorizzazioni ad applicazioni in esecuzione su istanze di Amazon EC2 nella Guida per l'utente IAM.

Per informazioni sull'utilizzo dei ruoli IAM, consulta Quando creare un ruolo IAM (invece di un utente) nella Guida per l'utente IAM.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e collegandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'operazione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'operazione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall'o dall' AWS API.

Policy basate su identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli nel tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scelta fra policy gestite e policy inline nella Guida per l'utente IAM.

Policy basate su risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi più comuni di policy basate su risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le azioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non puoi utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Liste di controllo degli accessi (ACL)

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano gli ACL. AWS WAF Per maggiori informazioni sulle ACL, consulta Panoramica delle liste di controllo degli accessi (ACL) nella Guida per gli sviluppatori di Amazon Simple Storage Service.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- Limiti delle autorizzazioni: un limite delle autorizzazioni è una funzionalità avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal`sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente IAM.
- Politiche di controllo dei servizi (SCP): le SCP sono politiche JSON che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in. AWS Organizations AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più Account AWS di proprietà dell'azienda. Se abiliti tutte le funzionalità in un'organizzazione, puoi applicare le policy di controllo dei servizi (SCP) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità negli account dei membri, inclusa ciascuna. Utente root dell'account AWS Per ulteriori informazioni su organizzazioni e policy SCP, consulta la pagina sulle Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations .
- Policy di sessione: le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

In che modo AWS HealthImaging funziona con IAM

Prima di utilizzare IAM per gestire l'accesso a HealthImaging, scopri con quali funzionalità IAM è disponibile l'uso HealthImaging.

Funzionalità IAM che puoi usare con AWS HealthImaging

Funzionalità IAM	HealthImaging supporto
Policy basate su identità	Sì
Policy basate su risorse	No
Azioni di policy	Sì
Risorse relative alle policy	Sì
Chiavi di condizione della policy (specifica del servizio)	Sì
Liste di controllo degli accessi (ACL)	No
ABAC (tag nelle policy)	Parziale
Credenziali temporanee	Sì
Autorizzazioni del principale	Sì
Ruoli di servizio	Sì
Ruoli collegati al servizio	No

Per avere una panoramica di alto livello su come HealthImaging e altri AWS servizi funzionano con la maggior parte delle funzionalità IAM, consulta AWS i servizi che funzionano con IAM nella IAM User Guide.

Politiche basate sull'identità per HealthImaging

Supporta le policy basate su identità	Sì

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente IAM.

Con le policy basate su identità di IAM, è possibile specificare quali operazioni e risorse sono consentite o respinte, nonché le condizioni in base alle quali le operazioni sono consentite o respinte. Non è possibile specificare l'entità principale in una policy basata sull'identità perché si applica all'utente o al ruolo a cui è associato. Per informazioni su tutti gli elementi utilizzabili in una policy JSON, consulta Guida di riferimento agli elementi delle policy JSON IAM nella Guida per l'utente di IAM.

Esempi di politiche basate sull'identità per HealthImaging

Per visualizzare esempi di politiche basate sull' HealthImaging identità, vedere. Esempi di policy basate sull'identità per AWS HealthImaging

Politiche basate sulle risorse all'interno HealthImaging

Supporta le policy basate su risorse	No

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi più comuni di policy basate su risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le azioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Per consentire l'accesso multi-account, puoi specificare un intero account o entità IAM in un altro account come principale in una policy basata sulle risorse. L'aggiunta di un principale multi-account a una policy basata sulle risorse rappresenta solo una parte della relazione di trust. Quando il principale e la risorsa sono diversi Account AWS, un amministratore IAM dell'account affidabile deve inoltre concedere all'entità principale (utente o ruolo) l'autorizzazione ad accedere alla risorsa. L'autorizzazione viene concessa collegando all'entità una policy basata sull'identità. Tuttavia, se una policy basata su risorse concede l'accesso a un principale nello stesso account, non sono richieste ulteriori policy basate su identità. Per ulteriori informazioni, consulta Cross Account Resource Access in IAM nella IAM User Guide.

Azioni politiche per HealthImaging

Supporta le operazioni di policy	Sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. Cioè, quale principale può eseguire azioni su quali risorse, e in quali condizioni.

L'elemento `Action`di una policy JSON descrive le azioni che è possibile utilizzare per consentire o negare l'accesso a un criterio. Le azioni politiche in genere hanno lo stesso nome dell'operazione AWS API associata. Ci sono alcune eccezioni, ad esempio le azioni di sola autorizzazione che non hanno un'operazione API corrispondente. Esistono anche alcune operazioni che richiedono più operazioni in una policy. Queste operazioni aggiuntive sono denominate operazioni dipendenti.

Includi le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di HealthImaging azioni, consulta Actions defined by AWS HealthImaging nel Service Authorization Reference.

Le azioni politiche in HealthImaging uso utilizzano il seguente prefisso prima dell'azione:

```
AWS
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [
      "AWS:action1",
      "AWS:action2"
```

```
        ]
```

Per visualizzare esempi di politiche HealthImaging basate sull'identità, vedere. Esempi di policy basate sull'identità per AWS HealthImaging

Risorse politiche per HealthImaging

Supporta le risorse di policy	Sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. Cioè, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento JSON `Resource`della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Le istruzioni devono includere un elemento `Resource`o un elemento `NotResource`. Come best practice, specifica una risorsa utilizzando il suo nome della risorsa Amazon (ARN). Puoi eseguire questa operazione per azioni che supportano un tipo di risorsa specifico, note come autorizzazioni a livello di risorsa.

Per le azioni che non supportano le autorizzazioni a livello di risorsa, ad esempio le operazioni di elenco, utilizza un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

Per visualizzare un elenco dei tipi di HealthImaging risorse e dei relativi ARN, consulta Tipi di risorse definiti da AWS HealthImaging nel Service Authorization Reference. Per sapere con quali azioni e risorse puoi usare un ARN, consulta Actions defined by AWS. HealthImaging

Per visualizzare esempi di politiche HealthImaging basate sull'identità, consulta. Esempi di policy basate sull'identità per AWS HealthImaging

Chiavi relative alle condizioni delle politiche per HealthImaging

Supporta le chiavi di condizione delle policy specifiche del servizio	Sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. Cioè, quale principale può eseguire azioni su quali risorse, e in quali condizioni.

L'elemento `Condition`(o blocco `Condition`) consente di specificare le condizioni in cui un'istruzione è in vigore. L'elemento `Condition`è facoltativo. Puoi compilare espressioni condizionali che utilizzano operatori di condizione, ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta.

Se specifichi più elementi `Condition`in un'istruzione o più chiavi in un singolo elemento `Condition`, questi vengono valutati da AWS utilizzando un'operazione `AND`logica. Se si specificano più valori per una singola chiave di condizione, AWS valuta la condizione utilizzando un'operazione logica. `OR` Tutte le condizioni devono essere soddisfatte prima che le autorizzazioni dell'istruzione vengano concesse.

Puoi anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, puoi autorizzare un utente IAM ad accedere a una risorsa solo se è stata taggata con il relativo nome utente IAM. Per ulteriori informazioni, consulta Elementi delle policy IAM: variabili e tag nella Guida per l'utente di IAM.

AWS supporta chiavi di condizione globali e chiavi di condizione specifiche del servizio. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di contesto delle condizioni AWS globali nella Guida per l'utente IAM.

Per visualizzare un elenco di chiavi di HealthImaging condizione, consulta Condition keys for AWS HealthImaging nel Service Authorization Reference. Per sapere con quali azioni e risorse puoi utilizzare una chiave di condizione, consulta Actions defined by AWS HealthImaging.

Per visualizzare esempi di politiche HealthImaging basate sull'identità, consulta. Esempi di policy basate sull'identità per AWS HealthImaging

ACL in HealthImaging

Supporta le ACL	No

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

RBAC con HealthImaging

Supporta RBAC	Sì

Il modello di autorizzazione tradizionale utilizzato in IAM è chiamato controllo dell'accesso basato sul ruolo (Role-Based Access Control, RBAC). RBAC definisce le autorizzazioni in base alla funzione lavorativa di una persona, nota anche al di fuori del ruolo. AWS Per ulteriori informazioni, consulta la sezione Confronto tra ABAC e il modello RBAC tradizionale nella IAM User Guide.

ABAC con HealthImaging

Supporta ABAC (tag nelle policy)	Parziale

Warning

ABAC non viene applicato tramite l'`SearchImageSets`azione API. Chiunque abbia accesso all'`SearchImageSets`azione può accedere a tutti i metadati per i set di immagini in un data store.

Note

I set di immagini sono una risorsa secondaria degli archivi di dati. Per utilizzare ABAC, un set di immagini deve avere lo stesso tag di un archivio dati. Per ulteriori informazioni, vedi Taggare le risorse con AWS HealthImaging.

Il controllo dell'accesso basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi. In AWS, questi attributi sono chiamati tag. Puoi allegare tag a entità IAM (utenti o ruoli) e a molte AWS risorse. L'assegnazione di tag alle entità e alle risorse è il primo passaggio di ABAC. In seguito, vengono progettate policy ABAC per consentire operazioni quando il tag dell'entità principale corrisponde al tag sulla risorsa a cui si sta provando ad accedere.

La strategia ABAC è utile in ambienti soggetti a una rapida crescita e aiuta in situazioni in cui la gestione delle policy diventa impegnativa.

Per controllare l'accesso basato su tag, fornisci informazioni sui tag nell'elemento condizione di una policy utilizzando le chiavi di condizione `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`o `aws:TagKeys`.

Se un servizio supporta tutte e tre le chiavi di condizione per ogni tipo di risorsa, il valore per il servizio è Yes (Sì). Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per ulteriori informazioni su ABAC, consulta Che cos'è ABAC? nella Guida per l'utente IAM. Per visualizzare un tutorial con i passaggi per l'impostazione di ABAC, consulta Utilizzo del controllo degli accessi basato su attributi (ABAC) nella Guida per l'utente di IAM.

Utilizzo di credenziali temporanee con HealthImaging

Supporta le credenziali temporanee	Sì

Alcuni Servizi AWS non funzionano quando si accede utilizzando credenziali temporanee. Per ulteriori informazioni, incluse quelle che Servizi AWS funzionano con credenziali temporanee, consulta la sezione relativa alla Servizi AWS compatibilità con IAM nella IAM User Guide.

Stai utilizzando credenziali temporanee se accedi AWS Management Console utilizzando qualsiasi metodo tranne nome utente e password. Ad esempio, quando accedi AWS utilizzando il link Single Sign-On (SSO) della tua azienda, tale processo crea automaticamente credenziali temporanee. Le credenziali temporanee vengono create in automatico anche quando accedi alla console come utente e poi cambi ruolo. Per ulteriori informazioni sullo scambio dei ruoli, consulta Cambio di un ruolo (console) nella Guida per l'utente IAM.

È possibile creare manualmente credenziali temporanee utilizzando l'API or. AWS CLI AWS È quindi possibile utilizzare tali credenziali temporanee per accedere. AWS AWS consiglia di generare dinamicamente credenziali temporanee anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza provvisorie in IAM.

Autorizzazioni principali multiservizio per HealthImaging

Supporta l'inoltro delle sessioni di accesso (FAS)	Sì

Quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Le policy concedono autorizzazioni a un'entità. Quando si utilizzano alcuni servizi, è possibile eseguire un'azione che attiva un'altra azione in un servizio diverso. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per vedere se un'azione richiede azioni dipendenti aggiuntive in una policy, consulta Azioni, risorse e chiavi di condizione per AWS HealthImaging nel Service Authorization Reference.

Ruoli di servizio per HealthImaging

Supporta i ruoli di servizio	Sì

Un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWSnella Guida per l'utente IAM.

> **Warning**
>
> La modifica delle autorizzazioni per un ruolo di servizio potrebbe compromettere HealthImaging la funzionalità. Modifica i ruoli di servizio solo quando viene HealthImaging fornita una guida in tal senso.

Ruoli collegati ai servizi per HealthImaging

Supporta i ruoli collegati ai servizi	No

Un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.

Per ulteriori informazioni su come creare e gestire i ruoli collegati ai servizi, consulta Servizi AWS supportati da IAM. Trova un servizio nella tabella che include un `Yes` nella colonna Service-linked role (Ruolo collegato ai servizi). Scegli il collegamento Sì per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Esempi di policy basate sull'identità per AWS HealthImaging

Per impostazione predefinita, gli utenti e i ruoli non sono autorizzati a creare o modificare risorse. HealthImaging Inoltre, non possono eseguire attività utilizzando AWS Management Console, AWS Command Line Interface (AWS CLI) o AWS l'API. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta Creazione di policy IAM nella Guida per l'utente di IAM.

Per i dettagli sulle azioni e sui tipi di risorse definiti da Awesome, incluso il formato degli ARN per ciascun tipo di risorsa, consulta Actions, Resources and Condition Keys for AWS Awesome nel Service Authorization Reference.

Argomenti

- Best practice per le policy
- Utilizzo della console di HealthImaging
- Consentire agli utenti di visualizzare le loro autorizzazioni

Best practice per le policy

Le politiche basate sull'identità determinano se qualcuno può creare, accedere o eliminare HealthImaging risorse nel tuo account. Queste azioni possono comportare costi aggiuntivi per l' Account AWS. Quando crei o modifichi policy basate su identità, segui queste linee guida e raccomandazioni:

- Inizia con le policy AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le politiche gestite che concedono le autorizzazioni per molti casi d'uso comuni.AWS Sono disponibili nel tuo. Account AWS Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai AWS clienti specifiche per i tuoi casi d'uso. Per ulteriori informazioni, consulta Policy gestite da AWSo Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente IAM.
- Applica le autorizzazioni con privilegio minimo: quando imposti le autorizzazioni con le policy IAM, concedi solo le autorizzazioni richieste per eseguire un'attività. Puoi farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come

autorizzazioni con privilegi minimi. Per ulteriori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta Policy e autorizzazioni in IAM nella Guida per l'utente IAM.

- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso: per limitare l'accesso a operazioni e risorse puoi aggiungere una condizione alle tue policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio AWS CloudFormation. Per ulteriori informazioni, consulta la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali: IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per ulteriori informazioni, consulta Convalida delle policy per IAM Access Analyzer nella Guida per l'utente IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungi le condizioni MFA alle policy. Per ulteriori informazioni, consulta Configurazione dell'accesso alle API protetto con MFA nella Guida per l'utente IAM.

Per maggiori informazioni sulle best practice in IAM, consulta Best practice di sicurezza in IAM nella Guida per l'utente di IAM.

Utilizzo della console di HealthImaging

Per accedere alla HealthImaging console AWS, devi disporre di un set minimo di autorizzazioni. Queste autorizzazioni devono consentirti di elencare e visualizzare i dettagli sulle HealthImaging risorse del tuo. Account AWS Se crei una policy basata sull'identità più restrittiva rispetto alle autorizzazioni minime richieste, la console non funzionerà nel modo previsto per le entità (utenti o ruoli) associate a tale policy.

Non è necessario consentire autorizzazioni minime per la console agli utenti che effettuano chiamate solo verso AWS CLI o l' AWS API. Al contrario, concedi l'accesso solo alle operazioni che corrispondono all'operazione API che stanno cercando di eseguire.

Per garantire che utenti e ruoli possano ancora utilizzare la HealthImaging console, allega anche la policy HealthImaging *ConsoleAccess* o la policy *ReadOnly* AWS gestita alle entità. Per ulteriori informazioni, consulta Aggiunta di autorizzazioni a un utente nella Guida per l'utente IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono cpllegate alla relativa identità utente. Questa politica include le autorizzazioni per completare questa azione sulla console o utilizzando l'API o a livello di codice. AWS CLI AWS

```
{
    "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": "*"
        }
    ]
```

```
}
```

AWSpolicy gestite per AWS HealthImaging

Una policy gestita da AWS è una policy autonoma creata e amministrata da AWS. Le policy gestite da AWS sono progettate per fornire autorizzazioni per molti casi d'uso comuni in modo da poter iniziare ad assegnare autorizzazioni a utenti, gruppi e ruoli.

Ricorda che le policy gestite da AWS potrebbero non concedere autorizzazioni con privilegi minimi per i tuoi casi d'uso specifici perché possono essere utilizzate da tutti i clienti AWS. Consigliamo pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente specifiche per i tuoi casi d'uso.

Non è possibile modificare le autorizzazioni definite nelle policy gestite da AWS. Se AWS aggiorna le autorizzazioni definite in una policy gestita da AWS, l'aggiornamento riguarda tutte le identità principali (utenti, gruppi e ruoli) a cui è collegata la policy. È molto probabile che AWS aggiorni una policy gestita da AWS quando viene lanciato un nuovo Servizio AWS o nuove operazioni API diventano disponibili per i servizi esistenti.

Per ulteriori informazioni, consultare Policy gestite da AWS nella Guida per l'utente di IAM.

Argomenti

- AWSpolitica gestita: AWSHealthImagingFullAccess
- AWSpolitica gestita: AWSHealthImagingReadOnlyAccess
- HealthImaging aggiornamenti alle politiche gestite AWS

AWSpolitica gestita: AWSHealthImagingFullAccess

È possibile allegare la policy `AWSHealthImagingFullAccess` alle identità IAM.

Questa politica concede l'autorizzazione amministrativa a tutte le HealthImaging azioni.

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

AWSpolitica gestita: AWSHealthImagingReadOnlyAccess

È possibile allegare la policy `AWSHealthImagingReadOnlyAccess` alle identità IAM.

Questa policy concede l'autorizzazione di sola lettura per azioni AWS specifiche. HealthImaging

```
{
    "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 aggiornamenti alle politiche gestite AWS

Visualizza i dettagli sugli aggiornamenti delle politiche AWS gestite HealthImaging da quando questo servizio ha iniziato a tenere traccia di queste modifiche. Per ricevere avvisi automatici sulle modifiche a questa pagina, iscriviti al feed RSS nella pagina Releases.

Modifica	Descrizione	Data
HealthImaging ha iniziato a tenere traccia delle modifiche	HealthImaging ha iniziato a tenere traccia delle modifiche per le sue politiche AWS gestite.	19 luglio 2023

Risoluzione dei problemi di HealthImaging identità e accesso AWS

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con HealthImaging IAM.

Argomenti

- Non sono autorizzato a eseguire alcuna azione in HealthImaging
- Non sono autorizzato a eseguire iam: PassRole
- Voglio consentire a persone esterne a me di accedere Account AWS alle mie HealthImaging risorse

Non sono autorizzato a eseguire alcuna azione in HealthImaging

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, le tue policy devono essere aggiornate per poter eseguire l'operazione.

L'errore di esempio seguente si verifica quando l'utente IAM `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *`my-example-widget`* fittizia ma non dispone di autorizzazioni `AWS:`*`GetWidget`* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 AWS:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa *`my-example-widget`* utilizzando l'azione `AWS:`*`GetWidget`*.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un messaggio di errore indicante che non sei autorizzato a eseguire l'`iam:PassRole`azione, le tue politiche devono essere aggiornate per consentirti di assegnare un ruolo a HealthImaging.

Alcuni Servizi AWS consentono di trasferire un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

Il seguente errore di esempio si verifica quando un utente IAM denominato `marymajor` tenta di utilizzare la console per eseguire un'azione in. HealthImaging Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per passare il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie HealthImaging risorse

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per servizi che supportano policy basate su risorse o liste di controllo degli accessi (ACL), utilizza tali policy per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se HealthImaging supporta queste funzionalità, consultaIn che modo AWS HealthImaging funziona con IAM.
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (Federazione delle identità) nella Guida per l'utente IAM.
- Per scoprire la differenza tra l'utilizzo di ruoli e politiche basate sulle risorse per l'accesso tra account diversi, consulta Cross Account Resource Access in IAM nella IAM User Guide.

Convalida della conformità per AWS HealthImaging

I revisori di terze parti valutano la sicurezza e la conformità di AWS nell' HealthImaging ambito di diversi programmi di AWS conformità. Infatti HealthImaging, questo include l'HIPAA.

Per un elenco dei servizi AWS coperti da programmi di conformità specifici, consulta Servizi AWS coperti dal programma di compliance. Per informazioni generali, consulta Programmi per la conformità di AWS.

È possibile scaricare i report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Download dei rapporti in AWS Artifact.

La tua responsabilità di conformità quando usi AWS HealthImaging è determinata dalla sensibilità dei tuoi dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWSfornisce le seguenti risorse per contribuire alla conformità:

- AWSSoluzioni per i partner: le guide di riferimento automatizzate all'implementazione per la sicurezza e la conformità illustrano le considerazioni relative all'architettura e forniscono i passaggi per implementare ambienti di base incentrati sulla sicurezza e la conformità. AWS
- Whitepaper sulla progettazione per la sicurezza HIPAA e la conformità: questo whitepaper descrive in che modo le aziende possono utilizzare AWS per creare applicazioni conformi ai requisiti HIPAA.
- GxP Systems on AWS: questo white paper fornisce informazioni su come AWS approcci alla conformità e alla sicurezza relative a GxP e fornisce indicazioni sull'utilizzo dei servizi nel contesto di GxP. AWS
- Risorse per la conformità di AWS: questa raccolta di workbook e guide potrebbe essere utile al tuo settore e alla tua posizione.
- Valutazione delle risorse con regole: AWS Config valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida e alle normative del settore.
- AWS Security Hub: Questo servizio AWS fornisce una visione completa dello stato di sicurezza all'interno di AWS che consente di verificare la conformità con gli standard e le best practice di sicurezza del settore.

Sicurezza dell'infrastruttura in AWS HealthImaging

In quanto servizio gestito, AWS HealthImaging è protetto dalle procedure di sicurezza di rete AWS globali descritte nel white paper Amazon Web Services: Overview of Security Processes.

Utilizzi chiamate API AWS pubblicate per accedere HealthImaging tramite la rete. I client devono supportare Transport Layer Security (TLS) 1.3 o versione successiva. I client devono, inoltre, supportare le suite di cifratura con PFS (Perfect Forward Secrecy), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate tramite un ID chiave di accesso e una chiave di accesso segreta associata a un principal IAM. È inoltre possibile utilizzare AWS Security Token Service(AWS STS) per generare credenziali di sicurezza temporanee per firmare le richieste.

Creazione di HealthImaging risorse AWS con AWS CloudFormation

AWS HealthImaging è integrato conAWS CloudFormation, un servizio che ti aiuta a modellare e configurare AWS le tue risorse in modo da poter dedicare meno tempo alla creazione e alla gestione

delle risorse e dell'infrastruttura. Crei un modello che descrive tutte le AWS risorse che desideri e fornisce AWS CloudFormation e configura tali risorse per te.

Quando lo utilizziAWS CloudFormation, puoi riutilizzare il modello per configurare le HealthImaging risorse in modo coerente e ripetuto. Basta descrivere le risorse una volta sola, dopodiché si può effettuare il provisioning di tali risorse quante volte si vuole in più Account AWS e regioni.

HealthImaging e modelli AWS CloudFormation

Per fornire e configurare le risorse HealthImaging e i servizi correlati, è necessario conoscere AWS CloudFormationi modelli. I modelli sono file di testo formattati in JSON o YAML. Questi modelli descrivono le risorse di cui intendi effettuare il provisioning negli stack AWS CloudFormation. Se non hai familiarità con JSON o YAML, puoi usare AWS CloudFormation Designer per iniziare a utilizzare i modelli AWS CloudFormation. Per ulteriori informazioni, consulta Che cos'è AWS CloudFormation Designer? nella Guida per l'utente di AWS CloudFormation.

AWS HealthImaging supporta la creazione di archivi dati conAWS CloudFormation. Per ulteriori informazioni, inclusi esempi di modelli JSON e YAML per il provisioning degli archivi di HealthImaging dati, consulta il riferimento ai tipi di HealthImaging risorse AWS nella User Guide. AWS CloudFormation

Ulteriori informazioni su AWS CloudFormation

Per ulteriori informazioni su AWS CloudFormation, consulta le seguenti risorse:

- AWS CloudFormation
- Guida per l'utente di AWS CloudFormation
- Documentazione di riferimento dell'API AWS CloudFormation
- Guida per l'utente dell'interfaccia a riga di comando di AWS CloudFormation

AWS HealthImaging e endpoint VPC di interfaccia ()AWS PrivateLink

Puoi stabilire una connessione privata tra il tuo VPC e creare un AWS HealthImaging endpoint VPC di interfaccia. Gli endpoint di interfaccia sono alimentati da AWS PrivateLink, una tecnologia che puoi utilizzare per accedere in modo privato alle HealthImaging API senza un gateway Internet, un dispositivo NAT, una connessione VPN o una connessione AWS Direct Connect. Le istanze nel tuo

VPC non necessitano di indirizzi IP pubblici per comunicare con HealthImaging le API. Il traffico tra il tuo VPC e HealthImaging non esce dalla rete Amazon.

Ogni endpoint dell'interfaccia è rappresentato da una o più interfacce di rete elastiche nelle tue sottoreti.

Per ulteriori informazioni, consulta Interface VPC endpoints (AWS PrivateLink) nella Amazon VPC User Guide.

Argomenti

- Considerazioni sugli endpoint HealthImaging VPC
- Creazione di un endpoint VPC di interfaccia per HealthImaging
- Creazione di una policy per gli endpoint VPC per HealthImaging

Considerazioni sugli endpoint HealthImaging VPC

Prima di configurare un endpoint VPC di interfaccia HealthImaging, assicurati di esaminare le proprietà e le limitazioni degli endpoint dell'interfaccia nella Amazon VPC User Guide.

HealthImaging supporta l'esecuzione di chiamate a tutte AWS HealthImaging le azioni dal tuo VPC.

Creazione di un endpoint VPC di interfaccia per HealthImaging

Puoi creare un endpoint VPC per il HealthImaging servizio utilizzando la console Amazon VPC o il (). AWS Command Line Interface AWS CLI Per ulteriori informazioni, consulta Creazione di un endpoint dell'interfaccia nella Guida per l'utente di Amazon VPC.

Crea endpoint VPC per HealthImaging utilizzare i seguenti nomi di servizio:

- com.amazonaws. *regione .medical-imaging*
- com.amazonaws. *regione.* runtime-medical-imaging
- com.amazonaws. *regione.* dicom-medical-imaging

> **Note**
>
> Il DNS privato deve essere abilitato per l'uso PrivateLink.

È possibile effettuare richieste API HealthImaging utilizzando il nome DNS predefinito per la regione, ad esempio. `medical-imaging.us-east-1.amazonaws.com`

Per ulteriori informazioni, consulta Accesso a un servizio tramite un endpoint dell'interfaccia in Guida per l'utente di Amazon VPC.

Creazione di una policy per gli endpoint VPC per HealthImaging

Puoi allegare una policy per gli endpoint al tuo endpoint VPC che controlla l'accesso a. HealthImaging Questa policy specifica le informazioni riportate di seguito:

- Il principale che può eseguire operazioni.
- Le azioni che possono essere eseguite
- Le risorse sui cui si possono eseguire le azioni

Per ulteriori informazioni, consultare Controllo degli accessi ai servizi con endpoint VPC in Guida per l'utente di Amazon VPC.

Esempio: policy degli endpoint VPC per le azioni HealthImaging

Di seguito è riportato un esempio di policy sugli endpoint per. HealthImaging Se associata a un endpoint, questa policy garantisce l'accesso alle HealthImaging azioni per tutti i principali utenti su tutte le risorse.

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\":\"*\"}]}"
```

Importazione tra più account per AWS HealthImaging

Con l'importazione tra account e regioni, puoi importare dati nel tuo HealthImaging data store da bucket Amazon S3 situati in altre regioni supportate. Puoi importare dati tra AWS account, account di proprietà di altre AWS Organizzazioni e da fonti di dati aperte come Imaging Data Commons (IDC) che si trovano nel Registry of Open Data su. AWS

HealthImaging I casi d'uso dell'importazione tra account e regioni includono:

- Prodotti SaaS per l'imaging medico che importano dati DICOM dagli account dei clienti
- Grandi organizzazioni che popolano un unico HealthImaging data store da molti bucket di input di Amazon S3
- I ricercatori condividono in modo sicuro i dati tra studi clinici multiistituzionali

Per utilizzare l'importazione tra più account

1. Il proprietario del bucket di input (source) di Amazon S3 deve concedere al proprietario del HealthImaging data store e le autorizzazioni. `s3:ListBucket` `s3:GetObject`
2. Il proprietario del HealthImaging data store deve aggiungere il bucket Amazon S3 al proprio IAM. `ImportJobDataAccessRole` Per informazioni, consulta Crea un ruolo IAM per l'importazione.
3. Il proprietario del HealthImaging data store deve fornire il bucket `inputOwnerAccountId`di input di Amazon S3 all'avvio del processo di importazione.

> **Note**
>
> Fornendo il`inputOwnerAccountId`, il proprietario del data store convalida l'input che il bucket Amazon S3 appartiene all'account specificato per mantenere la conformità agli standard di settore e mitigare i potenziali rischi per la sicurezza.

Il seguente esempio di `startDICOMImportJob` codice include il `inputOwnerAccountId` parametro opzionale, che può essere applicato a tutti, AWS CLI e gli esempi di codice SDK sono riportati nella sezione. Avvio di un processo di importazione

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 "";
```

```
}
```

Resilienza in AWS HealthImaging

L'infrastruttura globale dei servizi AWS è progettata attorno a regioni Regioni AWS e zone di disponibilità. Le regioni di Regioni AWS forniscono più zone di disponibilità fisicamente separate e isolate che sono connesse tramite reti altamente ridondanti, a bassa latenza e a velocità effettiva elevata. Con le Zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico tra zone di disponibilità senza interruzioni. Le Zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili, rispetto alle infrastrutture a data center singolo o multiplo.

Per ulteriori informazioni sulle Regioni AWS e le zone di disponibilità, consulta Infrastruttura globale di AWS.

Oltre all'infrastruttura AWS globale, AWS HealthImaging offre diverse funzionalità per supportare le esigenze di resilienza e backup dei dati.

Materiale HealthImaging di riferimento AWS

Il seguente materiale di riferimento è disponibile per AWS HealthImaging.

> **Note**
>
> Tutte HealthImaging le azioni e i tipi di dati si trovano in un riferimento separato. Per ulteriori informazioni, consulta l'AWS HealthImaging API Reference.

Argomenti

Supporto DICOM per AWS HealthImaging

AWS HealthImaging supporta elementi DICOM e sintassi di trasferimento specifici. Acquisisci familiarità con gli elementi di dati DICOM supportati a livello di paziente, studio e serie, poiché le chiavi dei HealthImaging metadati si basano su di essi. Prima di iniziare un'importazione, verificate che i dati di imaging medicale siano conformi alle sintassi di trasferimento supportate e ai vincoli HealthImaging degli elementi DICOM.

> **Note**
>
> Al momento AWS HealthImaging non supporta immagini di segmentazione binaria o dati in pixel di sequenze di immagini di icone.

Argomenti

- Normalizzazione dei metadati
- Sintassi di trasferimento supportate
- vincoli degli elementi DICOM
- vincoli dei metadati DICOM

Classi SOP supportate

Con AWS HealthImaging, puoi importare istanze DICOM P10 Service-Object Pair (SOP) codificate con qualsiasi UID di classe SOP, incluse quelle ritirate e private. Vengono inoltre preservati tutti gli attributi privati.

Normalizzazione dei metadati

Quando importi i dati DICOM P10 in AWS HealthImaging, questi vengono trasformati in set di immagini composti da metadati e frame di immagini (dati pixel). Durante il processo di trasformazione, le chiavi di HealthImaging metadati vengono generate in base a una versione specifica dello standard DICOM. HealthImaging attualmente genera e supporta chiavi di metadati basate sul dizionario dei dati DICOM PS3.6 2022b.

AWS HealthImaging supporta i seguenti elementi di dati DICOM a livello di paziente, studio e serie.

Elementi a livello di paziente

> **Note**
>
> Per una descrizione dettagliata di ogni elemento a livello di paziente, vedere il Registro degli elementi di dati DICOM.

AWS HealthImaging supporta i seguenti elementi a livello di paziente:

```
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
```

Elementi a livello di studio

> **Note**
>
> Per una descrizione dettagliata di ogni elemento del livello di studio, consulta il Registro degli elementi di dati DICOM.

AWS HealthImaging supporta i seguenti elementi a livello di studio:

```
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
```

Elementi a livello di serie

> **Note**
>
> Per una descrizione dettagliata di ogni elemento a livello di serie, consultate il Registro degli elementi di dati DICOM.

AWS HealthImaging supporta i seguenti elementi a livello di serie:

```
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
```

Sintassi di trasferimento supportate

AWS HealthImaging importa istanze SOP DICOM P10 codificate con le sintassi di trasferimento riportate nella tabella seguente. Oltre all'archiviazione dell'istanza SOP, HealthImaging transcodifica i frame di immagine (dati in pixel) in HTJ2K per istanze SOP codificate con le seguenti sintassi di trasferimento:

UID della sintassi di trasferimento	Nome della sintassi di trasferimento
1.2.840.100081.2	Implicit VR Endian: sintassi di trasferimento predefinita per DICOM
1.2.840.100081.2.1	Little Endian VR esplicito
1,2,840,10008,12.1.99	Little Endian VR esplicito sgonfio
1,2840.100081.2.2	Big Endian VR esplicito
1,2,840,10008,12,4,50	JPEG Baseline (Process 1): sintassi di trasferimento predefinita per la compressione di immagini JPEG a 8 bit con perdita di dati
1,2840.100081.2.4.51	JPEG Baseline (processi 2 e 4): sintassi di trasferimento predefinita per la compressione di immagini JPEG a 12 bit con perdita di dati (solo Process 4)
1,2840.100081.2.4.57	JPEG non gerarchico senza perdita di dati (processo 14)

UID della sintassi di trasferimento	Nome della sintassi di trasferimento
1,2,840,10008,12.4,70	JPEG Lossless, non gerarchica, previsione di primo ordine (Processi 14 [valore di selezione 1]): sintassi di trasferimento predefinita per la compressione delle immagini JPEG senza perdita di dati
1.2.840.100081.2.4.80	Compressione delle immagini senza perdita di dati JPEG-LS
1,2840.100081.2.4.81	Compressione delle immagini JPEG-LS con perdita (quasi senza perdita di dati)
1,2840.100081.2.4,90	Compressione delle immagini JPEG 2000 (solo senza perdita di dati)
1,2840.100081.2.4,91	Compressione delle immagini JPEG 2000
1,2840.100081.2.4.201	Compressione delle immagini JPEG 2000 ad alta produttività (solo senza perdita di dati)
1,2840.100081.2.4.202	JPEG 2000 ad alta produttività con opzioni di compressione delle immagini RPCL (solo senza perdita di dati)
1,2840.100081.2.4.203	Compressione delle immagini JPEG 2000 ad alta produttività
1.2.840.100081.2.5	RLE senza perdita

vincoli degli elementi DICOM

Quando si importano i dati di imaging medicale in AWS HealthImaging, i vincoli di lunghezza massima vengono applicati ai seguenti elementi DICOM. Per eseguire correttamente l'importazione, assicurati che i dati non superino i limiti di lunghezza massima.

Vincoli degli elementi DICOM durante l'importazione

HealthImaging parola chiave	parola chiave DICOM	Chiave DICOM	Limite di lunghezza
DICOM PatientName	Nome del paziente	(0010.0010)	minimo: 0, massimo: 256
ID del paziente DIC	ID del paziente	(0010.0020)	minimo: 0, massimo: 256
DICOM PatientBirthDate	Paziente BirthDate	(0010.0030)	minimo: 0, massimo: 18
DICOM PatientSex	PatientSex	(0010.0040)	minimo: 0, massimo: 16
UID DICOM StudyInstance	StudyInstanceUID	(0020.000 D)	min: 0, massimo: 64
DICOM StudyId	ID dello studio	(0020.0010)	minimo: 0, massimo: 16
DICOM StudyDescription	StudyDescription	(0008,1030)	minimo: 0, massimo: 64
DICOM NumberOfStudy RelatedSeries	Numero di serie relative allo studio	(0020, 1206)	minimo: 0, massimo: 10000
DICOM NumberOfStudy RelatedInstances	NumberOfStudyRelatedIstanze	(0020,1208)	minimo: 0, massimo: 10000
DICOM Accession Number	AccessionNumber	(008.0050)	minimo: 0, massimo: 256
DICOM StudyDate	StudyDate	(008.0020)	minimo: 0, massimo: 18
DICOM StudyTime	StudyTime	(008.0030)	minimo: 0, massimo: 28

vincoli dei metadati DICOM

Quando si utilizza `UpdateImageSetMetadata` per aggiornare gli attributi HealthImaging dei metadati, vengono applicati i seguenti vincoli DICOM.

- Non è possibile aggiornare o rimuovere gli attributi privati sugli attributi a livello di Patient/Study/Series/Instance a meno che il vincolo di aggiornamento non si applichi a entrambi e `updatableAttributes` `removableAttributes`
- Impossibile aggiornare i seguenti attributi HealthImaging generati da AWS: `SchemaVersion` `DatastoreIDImageSetID`,`PixelData`,`Checksum`,,`Width`,`Height`,`MinPixelValue`,`MaxPixelValue` `FrameSizeInBytes`
- Impossibile aggiornare i seguenti attributi DICOM:`Tag.PixelData`,,`Tag.StudyInstanceUID`,`Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID` `Tag.StudyID`
- Impossibile aggiornare gli attributi con VR type SQ (attributi annidati)
- Impossibile aggiornare gli attributi multivalore
- Impossibile aggiornare gli attributi con valori non compatibili con il tipo di attributo VR
- Impossibile aggiornare gli attributi che non sono considerati attributi validi secondo lo standard DICOM
- Impossibile aggiornare gli attributi tra i moduli. Ad esempio, se viene fornito un attributo a livello di paziente a livello di studio nella richiesta di payload del cliente, la richiesta può essere invalidata.
- Impossibile aggiornare gli attributi se il modulo di attributi associato non è presente nell'ambiente esistente. `ImageSetMetadata` Ad esempio, non è consentito aggiornare gli attributi per a `seriesInstanceUID` se il Series with non `seriesInstanceUID` è presente nei metadati del set di immagini esistente.

Verifica dei dati dei HealthImaging pixel AWS

Durante l'importazione, HealthImaging offre una verifica integrata dei dati dei pixel controllando lo stato di codifica e decodifica senza perdite di ogni immagine. Questa funzione assicura che le immagini decodificate utilizzando le librerie di decodifica HTJ2K corrispondano sempre alle immagini DICOM P10 originali importate. HealthImaging

- Il processo di onboarding delle immagini inizia quando un processo di importazione acquisisce lo stato di qualità dei pixel originale delle immagini DICOM P10 prima che vengano importate. Per

ogni immagine viene generato un Image Frame Resolution Checksum (IFRC) unico e immutabile utilizzando l'algoritmo CRC32. Viene calcolato un IFRC per livello di risoluzione per i dati dei pixel in ciascuna immagine. I valori del checksum IFRC sono presentati nel documento di metadati (`job-output-manifest.json`), ordinati in un elenco dalla risoluzione base alla risoluzione completa.

- Dopo che le immagini sono state importate in un archivio HealthImaging dati e trasformate in set di immagini, i fotogrammi delle immagini con codifica HTJ2K vengono immediatamente decodificati e vengono calcolati i nuovi IFRC. HealthImaging quindi confronta gli IFRC a piena risoluzione delle immagini originali con i nuovi IFRC dei fotogrammi di immagini importati per verificarne l'accuratezza.
- Una condizione di errore descrittiva corrispondente per immagine viene acquisita nell'import job output log () `job-output-manifest.json` per consentirne l'analisi e la verifica.

Per verificare i dati dei pixel

1. Dopo aver importato i dati di imaging medicale, visualizza il risultato descrittivo del successo (o della condizione di errore) del set di immagini per immagine registrato nel log di output del processo di importazione,. `job-output-manifest.json` Per ulteriori informazioni, consulta Comprendere i lavori di importazione.
2. I set di immagini sono composti da metadati e frame di immagini (dati in pixel). I metadati dei set di immagini contengono informazioni sui frame di immagini associati. Utilizzate l'`GetImageSetMetadata`azione per ottenere i metadati per un set di immagini. Per ulteriori informazioni, consulta Ottenere i metadati del set di immagini.
3. `PixelDataChecksumFromBaseToFullResolution`Contiene l'IFRC (checksum) per livello di risoluzione. Di seguito è riportato un esempio di output di metadati per l'IFRC generato come parte del processo di importazione e registrato in. `job-output-manifest.json`

```
"ImageFrames": [{
  "ID": "678906789067890123451234512345 12",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 128,
      "Height": 128,
      "Checksum": 2928338830
    },
    {
      "Width": 256,
```

```
      "Height": 256,
      "Checksum": 1362274918
    },
    {
      "Width": 512,
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
```

4. Per verificare i dati relativi ai pixel, accedi alla procedura di verifica dei dati Pixel GitHub e segui le istruzioni contenute nel `README.md` file per verificare in modo indipendente l'elaborazione delle immagini senza perdita di dati da parte dei vari Librerie di decodifica HTJ2K utenti di. HealthImaging Man mano che i dati vengono caricati progressivamente per livello di risoluzione, potete calcolare l'IFRC per i dati di input non elaborati e confrontarli con il valore IFRC fornito nei HealthImaging metadati per la stessa risoluzione per verificare i dati dei pixel.

Librerie di decodifica HTJ2K per AWS HealthImaging

Durante l'importazione, AWS HealthImaging codifica tutti i frame dell'immagine (dati dei pixel) in formato senza perdita di dati JPEG 2000 (HTJ2K) ad alta velocità per offrire una visualizzazione delle immagini costantemente veloce e un accesso universale alle funzionalità avanzate di HTJ2K. Poiché i frame delle immagini vengono codificati in HTJ2K durante l'importazione, devono essere decodificati prima di essere visualizzati in un visualizzatore di immagini.

> **Note**
>
> HTJ2K è definito nella parte 15 dello standard JPEG2000 (ISO/IEC 15444-15:2019). HTJ2K mantiene le funzionalità avanzate di JPEG2000 come la scalabilità della risoluzione, i distretti, la piastrellatura, l'elevata profondità di bit, i canali multipli e il supporto dello spazio colore.

Argomenti

- Librerie di decodifica HTJ2K
- Visualizzatori di immagini

Librerie di decodifica HTJ2K

A seconda del linguaggio di programmazione, consigliamo le seguenti librerie di decodifica per decodificare i frame delle immagini.

- NVIDIA NVJPEG2000: commerciale, accelerata da GPU
- Software Kakadu: commerciale, C++ con collegamenti Java e.NET
- OpenJPH: open source, C++ e WASM
- OpenJPEG — Open source, C/C++, Java
- openjphpy — Open source, Python
- pylibjpeg-openjpeg — Codice aperto, Python

Visualizzatori di immagini

È possibile visualizzare i riquadri delle immagini dopo averli decodificati. Le azioni HealthImaging dell'API AWS supportano una varietà di visualizzatori di immagini open source, tra cui:

- Fondazione Open Health Imaging (OHIF)
- Cornerstone.js

HealthImaging Endpoint e quote AWS

I seguenti argomenti contengono informazioni sugli endpoint e le quote dei HealthImaging servizi AWS.

Argomenti

- Endpoint di servizio
- Quote del servizio

Endpoint di servizio

Un endpoint di servizio è un URL che identifica un host e una porta come punto di ingresso per un servizio Web. Ogni richiesta di servizio Web include un endpoint. La maggior parte AWS dei servizi fornisce endpoint per regioni specifiche per consentire una connettività più rapida. La tabella seguente elenca gli endpoint del servizio per AWS HealthImaging.

Nome della regione	Regione	Endpoint	Protocollo	
US East (N. Virginia)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS	
US West (Oregon)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS	
Asia Pacific (Sydney)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS	
Europa (Irlanda)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS	

Se utilizzi richieste HTTP per chiamare HealthImaging azioni AWS, devi utilizzare endpoint diversi a seconda delle azioni chiamate. Il menu seguente elenca gli endpoint di servizio disponibili per le richieste HTTP e le azioni che supportano.

Azioni API supportate per le richieste HTTP

data store, import, tagging

Le seguenti azioni di archiviazione, importazione e etichettatura dei dati sono accessibili tramite endpoint:

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore
- ListDatastores

- DeleteDatastore
- Avvia DICOM ImportJob
- Ottieni DICOM ImportJob
- Elenca DICOM ImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

Le seguenti azioni del set di immagini sono accessibili tramite endpoint:

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata
- CopyImageSet

- DeleteImageSet

DICOMweb

HealthImaging offre una rappresentazione del servizio DICOMWeb Retrieve WADO-RS. Per ulteriori informazioni, consulta Ottenere un'istanza DICOM.

Il seguente servizio DICOMWeb è accessibile tramite endpoint:

```
https://dicom-medical-imaging.region.amazonaws.com
```

- getDICOM Instance

Quote del servizio

Le quote di servizio sono definite come il valore massimo per le risorse, le azioni e gli articoli presenti nell'account. AWS

> **Note**
>
> Per le quote regolabili, puoi richiedere un aumento della quota utilizzando la console Service Quotas. Per ulteriori informazioni, consulta Richiesta di un aumento di quota nella Guida per l'utente per Service Quotas.

La tabella seguente elenca le quote predefinite per AWS HealthImaging.

Nome	Predefinita	Adatta e	Descrizione
Numero massimo di CopyImageSet richieste simultanee per archivio dati	Ogni regione supportata: 100	Sì	Il numero massimo di CopyImageSet richieste simultanee per archivio dati nella regione corrente AWS
Numero massimo di DeleteImageSet richieste simultanee per archivio dati	Ogni regione supportata: 100	Sì	Il numero massimo di DeleteImageSet richieste

Nome	Predefinita	Adatta e	Descrizione
			simultanee per archivio dati nella regione corrente AWS
Numero massimo di UpdateImageSetMetadata richieste simultanee per archivio dati	Ogni regione supportata: 100	Sì	Il numero massimo di UpdateImageSetMetadata richieste simultanee per archivio dati nella regione corrente AWS
Numero massimo di processi di importazione simultanei per archivio dati	ap-southeast-2:20 Ogni altra regione supportata: 100	Sì	Il numero massimo di processi di importazione simultanei per archivio dati nella regione corrente AWS
Numero massimo di archivi dati	Ogni regione supportata: 10	Sì	Il numero massimo di archivi dati attivi nella AWS regione corrente
Numero massimo di copie ImageFrames consentite per richiesta CopyImageSet	Ogni regione supportata: 1.000	Sì	Il numero massimo di copie ImageFrames consentite per CopyImageSet richiesta nella regione corrente AWS
Numero massimo di file in un processo di importazione DICOM	Ogni regione supportata: 5.000	Sì	Il numero massimo di file in un processo di importazione DICOM nella regione corrente AWS

Nome	Predefinita	Adatta e	Descrizione
Numero massimo di cartelle annidate in un processo di importazione DICOM	Ogni regione supportata: 10.000	No	Il numero massimo di cartelle nidificate in un processo di importazione DICOM nella regione corrente AWS
Limite massimo di dimensione del payload (in KB) accettato da UpdateImageSetMetadata	Ogni regione supportata: 10 KB	Sì	Il limite massimo di dimensione del payload (in KB) accettato dalla UpdateImageSetMetadata regione corrente AWS
Dimensione massima (in GB) di tutti i file in un processo di importazione DICOM	Ogni regione supportata: 10 GB	No	La dimensione massima (in GB) di tutti i file in un processo di importazione DICOM nella regione corrente AWS
Dimensione massima (in GB) di ogni file DICOM P10 in un processo di importazione DICOM	Ogni regione supportata: 4 GB	No	La dimensione massima (in GB) di ogni file DICOM P10 nel processo di importazione DICOM nella regione corrente AWS
Limite massimo di dimensione (in MB) ImageSetMetadata per importazione, copia e UpdateImageSet	Ogni regione supportata: 50 MB	Sì	Il limite massimo di dimensione (in MB) ImageSetMetadata per importazione, copia e UpdateImageSet nella AWS regione corrente

Limiti di HealthImaging limitazione di AWS

Il tuo AWS account ha dei limiti di limitazione che si applicano alle azioni delle HealthImaging API AWS. Per tutte le azioni, viene `ThrottlingException` generato un errore se vengono superati i limiti di limitazione. Per ulteriori informazioni, consulta l'AWS HealthImaging API Reference.

> **Note**
>
> I limiti di limitazione sono regolabili per tutte le azioni HealthImaging dell'API. Per richiedere una regolazione del limite di limitazione, contatta il AWS Support Center.

La tabella seguente elenca i limiti di limitazione per le azioni delle HealthImaging API AWS.

Limiti di HealthImaging limitazione di AWS

Azione	Velocità di limitazione	Scoppio dell'acceleratore
CreateDatastore	0,085 tps	1 cucchiaino
GetDatastore	10 tps	20 cucchiaino
ListDatastores	5 tps	10 tps
DeleteDatastore	0,085 tps	1 cucchiaino
Avvia DICOM ImportJob	0,25 tps	1 cucchiaino
Ottieni DICOM ImportJob	25 suggerimenti	50 tps
Elenco DICOM ImportJobs	10 tps	20 consigli
SearchImageSets	25 tps	50 tps
GetImageSet	25 tps	50 tps
GetImageSetMetadata	50 tps	100 tps
GetImageFrame	1000 tps	2000 tps
Ottieni un'istanza DICOM*	50 suggerimenti	100 tps

Azione	Velocità di limitazione	Scoppio dell'acceleratore
ListImageSetVersions	25 tps	50 tps
UpdateImageSetMetadata	0,25 tps	1 cucchiaino
CopyImageSet	0,25 cucchiaino	1 cucchiaino
DeleteImageSet	0,25 cucchiaino	1 cucchiaino
TagResource	10 tps	20 cucchiaino
ListTagsForResource	10 tps	20 tps
UntagResource	10 tps	20 tps

*Rappresentazione di un servizio DICOMWeb

Progetti HealthImaging di esempio AWS

AWS HealthImaging fornisce i seguenti progetti di esempio su GitHub.

Inserimento di DICOM da locale ad AWS HealthImaging

Un progetto AWS serverless per l'implementazione di una soluzione IoT edge che riceve file DICOM da una fonte DICOM DIMSE (PACS, VNA, scanner CT) e li archivia in un bucket Amazon S3 sicuro. La soluzione indicizza i file DICOM in un database e mette in coda ogni serie DICOM da importare in AWS. HealthImaging È composta da un componente in esecuzione all'edge gestito da e da AWS IoT Greengrassuna pipeline di ingestione DICOM in esecuzione nel cloud. AWS

Proxy Tile Level Marker (TLM)

Un AWS Cloud Development Kit (AWS CDK)progetto per il recupero di frame di immagini HealthImaging da AWS utilizzando i tile level marker (TLM), una funzionalità di High-Throughput JPEG 2000 (HTJ2K). Ciò si traduce in tempi di recupero più rapidi con immagini a bassa risoluzione. I potenziali flussi di lavoro includono la generazione di miniature e il caricamento progressivo delle immagini.

CloudFront Amazon Delivery

Un progetto AWS serverless per la creazione di una CloudFront distribuzione Amazon con un endpoint HTTPS che memorizza nella cache (utilizzando GET) e fornisce frame di immagini dall'edge. Per impostazione predefinita, l'endpoint autentica le richieste con un token web JSON (JWT) di Amazon Cognito. Sia l'autenticazione che la firma delle richieste vengono eseguite all'edge utilizzando Lambda @Edge. Questo servizio è una funzionalità di Amazon CloudFront che ti consente di eseguire il codice più vicino agli utenti della tua applicazione, migliorando le prestazioni e riducendo la latenza. Non c'è alcuna infrastruttura da gestire.

Interfaccia utente AWS HealthImaging Viewer

Un AWS Amplifyprogetto per l'implementazione di un'interfaccia utente frontend con autenticazione backend con la quale è possibile visualizzare gli attributi dei metadati dei set di immagini e i frame di immagine (dati pixel) archiviati in HealthImaging AWS utilizzando la decodifica progressiva. Facoltativamente, puoi integrare il proxy Tile Level Marker (TLM) e/o i progetti Amazon CloudFront Delivery di cui sopra per caricare i frame di immagini utilizzando un metodo alternativo.

Per visualizzare altri progetti di esempio, consulta AWS HealthImaging Samples on GitHub.

Utilizzo HealthImaging con un AWS SDK

AWS I kit di sviluppo software (SDK) sono disponibili per molti linguaggi di programmazione più diffusi. Ogni SDK fornisce un'API, esempi di codice, e documentazione che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio preferito.

Documentazione sugli SDK	Esempi di codice
AWS SDK for C++	AWS SDK for C++ esempi di codice
AWS CLI	AWS CLI esempi di codice
AWS SDK for Go	AWS SDK for Go esempi di codice
AWS SDK for Java	AWS SDK for Java esempi di codice
AWS SDK for JavaScript	AWS SDK for JavaScript esempi di codice

Documentazione sugli SDK	Esempi di codice
SDK AWS for Kotlin	SDK AWS for Kotlin esempi di codice
AWS SDK for .NET	AWS SDK for .NET esempi di codice
AWS SDK for PHP	AWS SDK for PHP esempi di codice
AWS Tools for PowerShell	Strumenti per esempi di PowerShell codice
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) esempi di codice
AWS SDK for Ruby	AWS SDK for Ruby esempi di codice
AWS SDK for Rust	AWS SDK for Rust esempi di codice
SDK AWS per SAP ABAP	SDK AWS per SAP ABAP esempi di codice
SDK AWS per Swift	SDK AWS per Swift esempi di codice

Esempio di disponibilità

Non riesci a trovare quello che ti serve? Richiedi un esempio di codice utilizzando il link Provide feedback (Fornisci un feedback) nella parte inferiore di questa pagina.

HealthImaging Versioni di AWS

La tabella seguente mostra quando sono stati rilasciati funzionalità e aggiornamenti per il HealthImaging servizio e la documentazione AWS.

Modifica	Descrizione	Data

Supporto migliorato per le importazioni di dati DICOM non standard	HealthImaging supporta le importazioni di dati che includono deviazioni dallo standard DICOM. Per ulteriori informazioni, consulta vincoli degli elementi DICOM.
	• I seguenti elementi di dati DICOM possono avere una lunghezza massima di 256 caratteri:
	  • `Patient's Name (0010,0010)`
	  • `Patient ID (0010,0020)`
	  • `Accession Number (0008,0050)`
	• Le seguenti variazioni di sintassi sono consentit e per`Study Instance UID`,`Series Instance UID`,`Treatment Session UID`, `Manufactu rer’s Device Class UIDDevice UID`, e: `Acquisition UID`
		28 giugno 2024

	• Il primo elemento di qualsiasi UID può essere zero • Gli UID possono iniziare con uno o più zeri iniziali • Gli UID possono avere una lunghezza massima di 256 caratteri	
Notifiche eventi	HealthImaging si integra con Amazon EventBridge per supportare applicazioni basate sugli eventi. Per ulteriori informazioni, consulta Usare Amazon EventBridge con HealthImaging.	5 giugno 2024
`GetDICOMInstance` per la restituzione di dati DICOM	HealthImaging fornisce il `GetDICOMInstance` servizio per restituire i dati (`.dcm`file) DICOM Part 10 per un determinato set di immagini. Per ulteriori informazioni, consulta Ottenere un'istanza DICOM.	15 maggio 2024
Importazione da più account	HealthImaging supporta l'importazione di dati da bucket Amazon S3 situati in altre regioni supportate. Per ulteriori informazioni, consulta Importazione tra più account per AWS HealthImaging.	15 maggio 2024

Miglioramenti della ricerca per i set di immagini	HealthImaging `SearchImageSets` action supporta i seguenti miglioramenti della ricerca. Per ulteriori informazioni, consulta Ricerca di set di immagini. • Supporto aggiuntivo per la ricerca su e `UpdatedAt` `SeriesInstanceUID` • Cerca tra l'ora di inizio e l'ora di fine • Ordina i risultati della ricerca per `Ascending` o `Descending` • I parametri della serie DICOM vengono restituiti nelle risposte	3 aprile 2024
La dimensione massima dei file per le importazioni è aumentata	HealthImaging supporta una dimensione massima di 4 GB per ogni file DICOM P10 in un processo di importazione. Per ulteriori informazioni, consulta Quote del servizio.	6 marzo 2024

Sintassi di trasferimento per JPEG Lossless e HTJ2K	HealthImaging supporta le seguenti sintassi di trasferimento per l'importazione di lavori. Per ulteriori informazioni, consulta Sintassi di trasferimento supportate. • 1.2.840.10008.1.2.4.57 — JPEG Lossless non gerarchico (processo 14) • 1.2.840.10008.1.2.4.201 — Compressione di immagini JPEG 2000 ad alto rendimento (solo senza perdita di dati) • 1.2.840.10008.1.2.4.202 — JPEG 2000 ad alto rendimento con opzioni di compressione delle immagini RPCL (solo senza perdita di dati) • 1.2.840.10008.1.2.4.203 — Compressione di immagini JPEG 2000 ad alto rendimento	16 febbraio 2024
Esempi di codice testati	HealthImaging la documentazione fornisce esempi di codice testati AWS CLI e AWS SDK per Python JavaScript, Java e C++. Per ulteriori informazioni, consulta Esempi di codice per l' HealthImaging utilizzo degli AWS SDK.	19 dicembre 2023

Il numero massimo di file per le importazioni è aumentato	HealthImaging supporta fino a 5.000 file per un singolo processo di importazione. Per ulteriori informazioni, consulta Quote del servizio.	19 dicembre 2023
Cartelle annidate per le importazioni	HealthImaging supporta fino a 10.000 cartelle nidificate per un singolo processo di importazione. Per ulteriori informazioni, consulta Quote del servizio.	1 dicembre 2023
Importazioni più rapide	HealthImaging fornisce importazioni 20 volte più veloci in tutte le regioni supportat e. Per ulteriori informazioni, consulta Endpoint di servizio.	1 dicembre 2023
CloudFormation supporto	HealthImaging supporta Infrastructure as Code (IaC) per il provisioning degli archivi dati. Per ulteriori informazi oni, consulta Creazione di HealthImaging risorse AWS con AWS CloudFormation.	21 settembre 2023
Disponibilità generale	AWS HealthImaging è disponibile per tutti i clienti nelle regioni Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon), Europa (Irlanda) e Asia Pacifico (Sydney).	26 luglio 2023

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.