aws

Guide du développeur

AWS HealthImaging

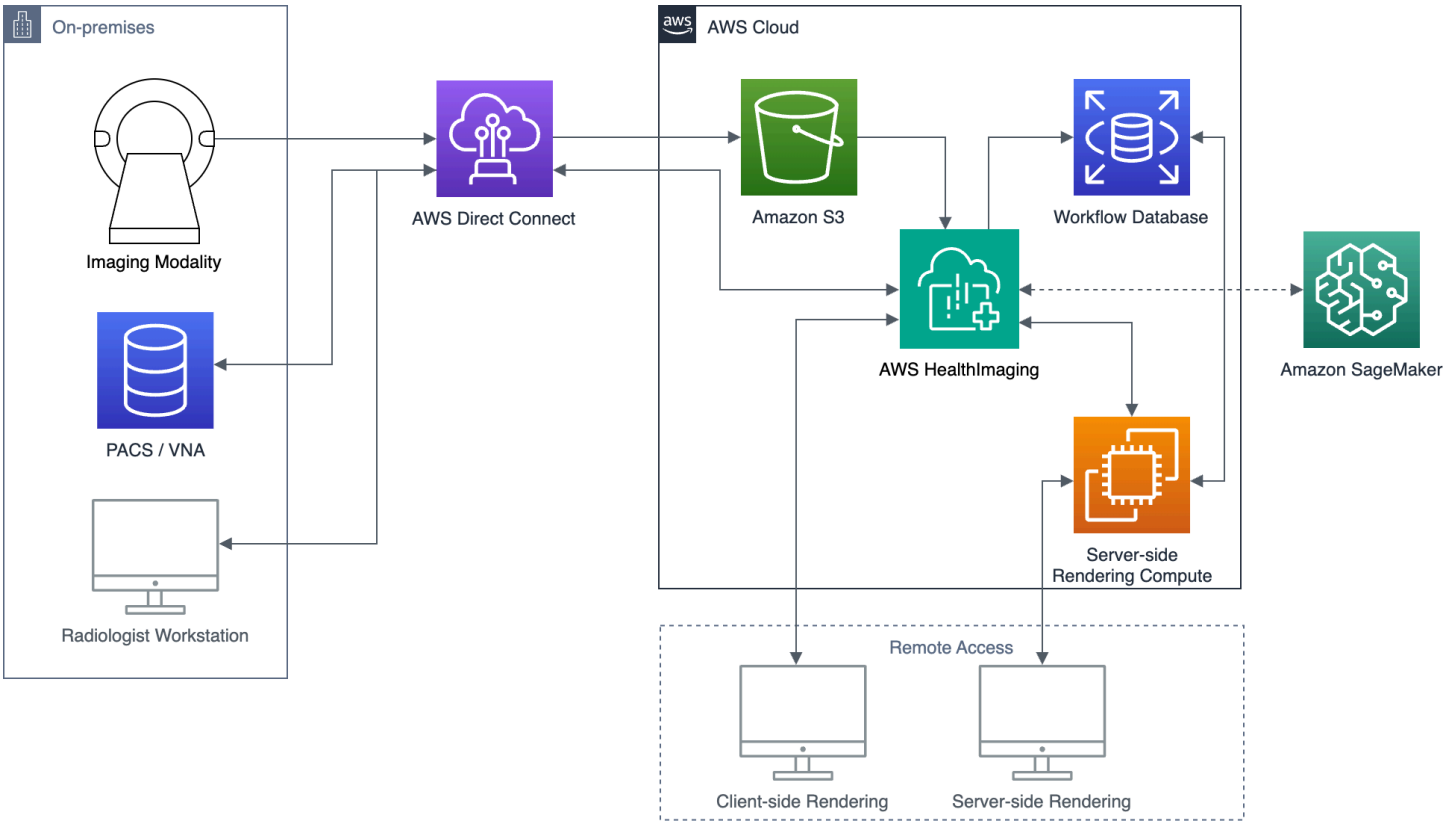
AWS HealthImaging: Guide du développeur

Table of Contents

Qu'est-ce qu'AWS HealthImaging ?

AWS HealthImaging est un service éligible à la loi HIPAA qui permet aux prestataires de soins de santé, aux organisations du secteur des sciences de la vie et à leurs partenaires logiciels de stocker, d'analyser et de partager des images médicales dans le cloud à l'échelle du pétaoctet. HealthImagingles cas d'utilisation incluent :

- Imagerie d'entreprise : stockez et diffusez vos données d'imagerie médicale directement depuis AWS le cloud tout en préservant des performances à faible latence et une haute disponibilité.
- Archivage d'images à long terme — Réduisez les coûts d'archivage d'images à long terme tout en conservant un accès à la récupération des images en moins d'une seconde.
- Développement IA/ML — Exécutez des inférences basées sur l'intelligence artificielle et l'apprentissage automatique (AI/ML) sur vos archives d'imagerie avec l'aide d'autres outils et services.
- Analyse multimodale — Combinez vos données d'imagerie clinique avec AWS HealthLake (données de santé) et AWS HealthOmics (données omiques) pour obtenir des informations utiles à la médecine de précision.



AWS HealthImaging fournit un accès aux données d'image (par exemple X-Ray, CT, IRM, Ultrasound) afin que les applications d'imagerie médicale créées dans le cloud puissent atteindre des performances auparavant réservées aux applications sur site. Vous réduisez ainsi les coûts d'infrastructure en exécutant vos applications d'imagerie médicale à grande échelle à partir d'une copie unique et fiable de chaque image médicale intégrée. HealthImaging AWS Cloud

Rubriques

Avis important

AWS HealthImaging ne remplace pas les conseils, diagnostics ou traitements médicaux professionnels et n'est pas destiné à guérir, traiter, atténuer, prévenir ou diagnostiquer une maladie ou un problème de santé. Il vous incombe de mettre en place un examen humain dans le cadre de toute utilisation d'AWS HealthImaging, y compris en association avec un produit tiers destiné à éclairer la prise de décisions cliniques. AWS ne HealthImaging doit être utilisé dans le cadre de soins aux patients ou de scénarios cliniques qu'après examen par des professionnels de santé qualifiés faisant preuve de bon jugement médical.

Fonctionnalités d'AWS HealthImaging

AWS HealthImaging fournit les fonctionnalités suivantes.

Métadonnées DICOM conviviales pour les développeurs

AWS HealthImaging simplifie le développement d'applications en renvoyant les métadonnées DICOM dans un format convivial pour les développeurs. Après avoir importé vos données d'imagerie, les attributs de métadonnées individuels sont accessibles à l'aide de mots clés conviviaux plutôt que de nombres hexadécimaux de groupe/élément inconnus. Les éléments

DICOM au niveau du patient, de l'étude et de la série sont normalisés, ce qui évite aux développeurs d'applications de devoir gérer les incohérences entre les instances SOP. En outre, les valeurs des attributs de métadonnées sont directement accessibles dans les types d'exécution natifs.

Décodage d'image accéléré par carte SIM

AWS HealthImaging renvoie des images (données en pixels) codées avec le codec de compression d'image avancé JPEG 2000 (HTJ2K) à haut débit. Le HTJ2K tire parti de la technologie SIMD (Single Instruction Multiple Data) des processeurs modernes pour offrir de nouveaux niveaux de performance. Le HTJ2K est un ordre de grandeur plus rapide que le JPEG2000 et au moins deux fois plus rapide que toutes les autres syntaxes de transfert DICOM. Le WASM-SIMD peut être utilisé pour réduire cette vitesse extrême à un encombrement nul pour les internautes.

Vérification des données des pixels

AWS HealthImaging fournit une vérification intégrée des données de pixels en vérifiant l'état de codage et de décodage sans perte de chaque image lors de l'importation. Pour plus d’informations, consultez Vérification des données des pixels.

Des performances de pointe

AWS HealthImaging définit une nouvelle norme en matière de performances de chargement d'images grâce à son encodage efficace des métadonnées, à sa compression sans perte et à son accès aux données en résolution progressive. Le codage efficace des métadonnées permet aux visionneuses d'images et aux algorithmes d'intelligence artificielle de comprendre le contenu d'une étude DICOM sans avoir à charger les données d'image. Les images se chargent plus rapidement sans compromettre la qualité de l'image grâce à une compression d'image avancée. La résolution progressive permet un chargement encore plus rapide des images pour les miniatures, les régions d'intérêt et les appareils mobiles basse résolution.

Importations DICOM évolutives

HealthImaging Les importations AWS tirent parti des technologies cloud natives modernes pour importer plusieurs études DICOM en parallèle. Les archives historiques peuvent être importées rapidement sans affecter les charges de travail cliniques liées aux nouvelles données. Pour plus d'informations sur les instances SOP prises en charge et les syntaxes de transfert, consultez. Support DICOM

AWS Services connexes

AWS HealthImaging offre une intégration étroite avec les autres AWS services. Il est utile de connaître les services suivants pour en tirer pleinement parti HealthImaging.

- AWS Identity and Access Management— Utilisez IAM pour gérer en toute sécurité les identités et l'accès aux HealthImaging ressources.
- Amazon Simple Storage Service : utilisez Amazon S3 comme zone intermédiaire pour importer des données DICOM dans HealthImaging.
- Amazon CloudWatch — CloudWatch À utiliser pour observer et surveiller les HealthImaging ressources.
- AWS CloudTrail— CloudTrail À utiliser pour suivre HealthImaging l'activité des utilisateurs et l'utilisation de l'API.
- AWS CloudFormation— AWS CloudFormation À utiliser pour implémenter des modèles d'infrastructure sous forme de code (IaC) dans lesquels créer des ressources HealthImaging.
- AWS PrivateLink— Utilisez Amazon VPC pour établir une connectivité entre Amazon Virtual Private Cloud HealthImaging et Amazon sans exposer les données à Internet.
- Amazon EventBridge — EventBridge À utiliser pour créer des applications évolutives axées sur les événements en créant des règles qui acheminent les HealthImaging événements vers des cibles.

Accès à AWS HealthImaging

Vous pouvez accéder à AWS à l' HealthImaging aide des kits de développement logiciel (SDK) AWS Command Line Interface et des AWS kits de développement logiciel (SDK). AWS Management Console Ce guide fournit des instructions de procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK.

AWS Management Console

> AWS Management Console Fournit une interface utilisateur basée sur le Web pour la gestion HealthImaging et les ressources associées. Si vous avez créé un AWS compte, vous pouvez vous connecter à la HealthImaging console.

AWS Command Line Interface (AWS CLI)

AWS CLI Il fournit des commandes pour un large éventail de AWS produits et est pris en charge sous Windows, Mac et Linux. Pour plus d'informations, consultez le Guide de l'utilisateur AWS Command Line Interface .

AWS SDK

AWS Les SDK fournissent des bibliothèques, des exemples de code et d'autres ressources aux développeurs de logiciels. Ces bibliothèques fournissent des fonctions de base qui automatisent des tâches telles que la signature cryptographique de vos demandes, les nouvelles tentatives et la gestion des réponses aux erreurs. Pour de plus amples informations, veuillez consulter Outils pour créer sur AWS.

Requêtes HTTP

Vous pouvez appeler HealthImaging des actions à l'aide de requêtes HTTP, mais vous devez spécifier différents points de terminaison en fonction du type d'action utilisé. Pour plus d'informations, consultez Actions d'API prises en charge pour les requêtes HTTP.

Éligibilité à la loi HIPAA et sécurité des données

Il s'agit d'un service admissible en vertu de la loi HIPAA. Pour plus d'informations sur AWS le Health Insurance Portability and Accountability Act des États-Unis de 1996 (HIPAA) et sur l'utilisation de AWS services pour traiter, stocker et transmettre des informations de santé protégées (PHI), consultez la section Présentation de la HIPAA.

Les connexions HealthImaging contenant des PHI et des informations personnelles identifiables (PII) doivent être cryptées. Par défaut, toutes les connexions doivent HealthImaging utiliser le protocole HTTPS sur TLS. HealthImaging stocke le contenu crypté des clients et fonctionne selon le modèle de responsabilitéAWS partagée.

Pour plus d'informations sur la conformité, consultezValidation de conformité pour AWS HealthImaging.

Tarification

HealthImaging vous aide à automatiser la gestion du cycle de vie des données cliniques grâce à une hiérarchisation intelligente. Pour plus d'informations, consultez Comprendre les niveaux de stockage.

Pour obtenir des informations générales sur les tarifs, consultez la section HealthImaging Tarification AWS. Pour estimer les coûts, utilisez le calculateur de HealthImaging prix AWS.

Commencer à utiliser AWS HealthImaging

Pour commencer à utiliser AWS HealthImaging, configurez un AWS compte et créez un AWS Identity and Access Management utilisateur. Pour utiliser le AWS CLIou les AWS SDK, vous devez les installer et les configurer.

Après avoir appris HealthImaging les concepts et la configuration, un court didacticiel avec des exemples de code est disponible pour vous aider à démarrer.

Rubriques

- HealthImaging Concepts AWS
- Configuration d'AWS HealthImaging
- HealthImaging Tutoriel AWS

HealthImaging Concepts AWS

La terminologie et les concepts suivants sont essentiels à votre compréhension et à votre utilisation d'AWS HealthImaging.

Concepts

- Banque de données
- Ensemble d'images
- Metadonnées
- Cadre d'image

Banque de données

Un magasin de données est un référentiel de données d'imagerie médicale qui se trouve dans un répertoire uniqueRégion AWS. Un AWS compte peut avoir zéro ou plusieurs banques de données. Un magasin de données possède sa propre clé de AWS KMS chiffrement, de sorte que les données d'un magasin de données peuvent être isolées physiquement et logiquement des données des autres magasins de données. Les magasins de données prennent en charge le contrôle d'accès à l'aide des rôles IAM, des autorisations et du contrôle d'accès basé sur les attributs.

Pour plus d'informations, consultez Gestion des magasins de données et Comprendre les niveaux de stockage.

Ensemble d'images

Un ensemble d'images est un AWS concept qui définit un mécanisme de regroupement abstrait pour optimiser les données d'imagerie médicale associées. Lorsque vous importez vos données d'imagerie DICOM P10 dans un magasin de HealthImaging données AWS, elles sont transformées en ensembles d'images composés de métadonnées et de cadres d'image (données en pixels). L'importation de données DICOM P10 produit des ensembles d'images contenant des métadonnées DICOM et des cadres d'image pour une ou plusieurs instances de paire Service-Object (SOP) de la même série DICOM.

Pour plus d'informations, consultez Importation de données d'imagerie et Comprendre les ensembles d'images.

Metadonnées

Les métadonnées sont les attributs autres que les pixels qui existent dans un ensemble d'images. Pour DICOM, cela inclut les données démographiques des patients, les détails des procédures et d'autres paramètres spécifiques à l'acquisition. AWS HealthImaging sépare le jeu d'images en métadonnées et en cadres d'image (données en pixels) afin que les applications puissent y accéder rapidement. Cela est utile pour les visionneuses d'images, les analyses et les cas d'utilisation de l'IA et du ML qui ne nécessitent pas de données en pixels. Les données DICOM se normalisent au niveau du patient, de l'étude et de la série, éliminant ainsi les incohérences. Cela simplifie l'utilisation des données, accroît la sécurité et améliore les performances d'accès.

Pour plus d'informations, consultez Obtenir les métadonnées d'un ensemble d'images et Normalisation des métadonnées.

Cadre d'image

Une trame d'image correspond aux données de pixels qui existent dans un ensemble d'images pour constituer une image médicale en 2D. Lors de l'importation, AWS HealthImaging code toutes les images au format JPEG 2000 à haut débit (HTJ2K). Par conséquent, les images doivent être décodées avant d'être visionnées.

Pour plus d'informations, consultez Obtenir les données en pixels d'un ensemble d'images et bibliothèques de décodage HTJ2K.

Configuration d'AWS HealthImaging

Vous devez configurer votre AWS environnement avant d'utiliser AWS HealthImaging. Les rubriques suivantes sont des prérequis pour le didacticiel présenté dans la section suivante.

Rubriques

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez https://portal.aws.amazon.com/billing/signup.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. Pour des raisons de sécurité, attribuez un accès administratif à un utilisateur et utilisez uniquement l'utilisateur root pour effectuer les tâches nécessitant un accès utilisateur root.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. Vous pouvez afficher l'activité en cours de votre compte et gérer votre compte à tout moment en accédant à https://aws.amazon.com/ et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Après vous être inscrit à un Compte AWS, sécurisez Utilisateur racine d'un compte AWS AWS IAM Identity Center, activez et créez un utilisateur administratif afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, voir Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connectez-vous en tant qu'utilisateur disposant d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribuer l'accès à des utilisateurs supplémentaires

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme aux meilleures pratiques en matière d'application des autorisations du moindre privilège.

   Pour obtenir des instructions, voir Création d'un ensemble d'autorisations dans le guide de AWS IAM Identity Center l'utilisateur.

2. Affectez des utilisateurs à un groupe, puis attribuez un accès d'authentification unique au groupe.

   Pour obtenir des instructions, voir Ajouter des groupes dans le guide de AWS IAM Identity Center l'utilisateur.

Création de compartiments S3

Pour importer des données DICOM P10 dans AWS HealthImaging, deux compartiments Amazon S3 sont recommandés. Le compartiment d'entrée Amazon S3 stocke les données DICOM P10 à importer et les HealthImaging lit depuis ce compartiment. Le compartiment de sortie Amazon S3 stocke les résultats du traitement de la tâche d'importation et HealthImaging écrit dans ce compartiment. Pour une représentation visuelle de cela, voir le schéma à l'adresseComprendre les tâches d'importation.

> **Note**
>
> En raison de la politique AWS Identity and Access Management (IAM), les noms de vos compartiments Amazon S3 doivent être uniques. Pour plus d'informations, veuillez consulter la section Règles de dénomination de compartiment dans le Guide de l'utilisateur Amazon Simple Storage Service.

Dans le cadre de ce guide, nous indiquons les compartiments d'entrée et de sortie Amazon S3 suivants dans le rôle IAM pour l'importation.

- Seau d'entrée : `arn:aws:s3:::medical-imaging-dicom-input`
- Seau de sortie : `arn:aws:s3:::medical-imaging-output`

Pour plus d'informations, consultez la section Création d'un compartiment dans le guide de l'utilisateur Amazon S3.

Création d'un magasin de données

Lorsque vous importez vos données d'imagerie médicale, le magasin de HealthImaging données AWS contient les résultats de vos fichiers DICOM P10 transformés, appelés ensembles d'images. Pour une représentation visuelle de cela, voir le schéma à l'adresseComprendre les tâches d'importation.

> **Tip**
>
> A `datastoreID` est généré lorsque vous créez un magasin de données. Vous devez utiliser le `datastoreID` lorsque vous complétez le formulaire trust relationship d'importation plus loin dans cette section.

Pour créer un magasin de données, voirCréation d'un magasin de données.

Création d'un utilisateur IAM avec une autorisation d'accès HealthImaging complète

> **Bonne pratique**
>
> Nous vous suggérons de créer des utilisateurs IAM distincts pour différents besoins tels que l'importation, l'accès aux données et la gestion des données. Cela correspond à l'octroi d'un accès au moindre privilège dans le AWS Well-Architected Framework.
> Pour les besoins du didacticiel de la section suivante, vous utiliserez un seul utilisateur IAM.

Pour créer un utilisateur IAM

1. Suivez les instructions relatives à la création d'un utilisateur IAM dans votre AWS compte dans le guide de l'utilisateur IAM. Envisagez de nommer l'utilisateur `ahiadmin` (ou un nom similaire) à des fins de clarification.
2. Attribuez la politique `AWSHealthImagingFullAccess` gérée à l'utilisateur IAM. Pour plus d'informations, consultez AWSpolitique gérée : AWSHealthImagingFullAccess.

> **Note**
>
> Les autorisations IAM peuvent être restreintes. Pour plus d'informations, consultez AWSpolitiques gérées pour AWS HealthImaging.

Création d'un rôle IAM pour l'importation

> **Note**
>
> Les instructions suivantes font référence à un rôle AWS Identity and Access Management (IAM) qui accorde un accès en lecture et en écriture aux compartiments Amazon S3 pour importer vos données DICOM. Bien que le rôle soit obligatoire pour le didacticiel de la section suivante, nous vous recommandons d'ajouter des autorisations IAM aux utilisateurs, aux groupes et aux rôles qui les utilisentAWSpolitiques gérées pour AWS HealthImaging, car elles sont plus faciles à utiliser que de rédiger vous-même des politiques.

Un rôle IAM est une identité IAM que vous pouvez créer dans votre compte et qui dispose d'autorisations spécifiques. Pour démarrer une tâche d'importation, le rôle IAM qui appelle l'`StartDICOMImportJob`action doit être associé à une politique utilisateur qui accorde l'accès aux compartiments Amazon S3 utilisés pour lire vos données DICOM P10 et stocker les résultats du traitement de la tâche d'importation. Il doit également se voir attribuer une relation de confiance (politique) qui permet HealthImaging à AWS d'assumer ce rôle.

Pour créer un rôle IAM à des fins d'importation

1. À l'aide de la console IAM, créez un rôle nomméImportJobDataAccessRole. Vous utiliserez ce rôle pour le didacticiel de la section suivante. Pour plus d'informations, consultez Création de rôles IAM dans le Guide de l'utilisateur IAM.

   > **Tip**
   >
   > Dans le cadre de ce guide, les exemples de code présentés font Démarrage d'une tâche d'importation référence au rôle `ImportJobDataAccessRole` IAM.

2. Associez une politique d'autorisation IAM au rôle IAM. Cette politique d'autorisation accorde l'accès aux compartiments d'entrée et de sortie Amazon S3. Associez la politique d'autorisation suivante au rôle `ImportJobDataAccessRole` IAM.

```
{
    "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. Attachez la relation de confiance (politique) suivante au rôle `ImportJobDataAccessRole` IAM. La politique de confiance exige que `datastoreId` ce qui a été généré lorsque vous avez terminé la sectionCréation d'un magasin de données. Le didacticiel suivant cette rubrique part du principe que vous utilisez un seul magasin de HealthImaging données AWS, mais avec des

compartiments Amazon S3, des rôles IAM et des politiques de confiance spécifiques au magasin de données.

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

Pour en savoir plus sur la création et l'utilisation de politiques IAM avec AWS HealthImaging, consultezIdentity and Access Management pour AWS HealthImaging.

Pour en savoir plus sur les rôles IAM en général, consultez la section Rôles IAM dans le Guide de l'utilisateur IAM. Pour en savoir plus sur les politiques et autorisations IAM en général, consultez la section Politiques et autorisations IAM dans le guide de l'utilisateur IAM.

Installez le AWS CLI (facultatif)

La procédure suivante est requise si vous utilisez le AWS Command Line Interface. Si vous utilisez le AWS Management Console ou les AWS SDK, vous pouvez ignorer la procédure suivante.

Pour configurer le AWS CLI

1. Téléchargez et configurez l’interface AWS CLI. Pour obtenir des instructions, consultez les rubriques suivantes dans le Guide de l’utilisateur de l’AWS Command Line Interface :

- Installation ou mise à jour de la dernière version du AWS CLI
- Commencer à utiliser le AWS CLI

2. Dans le AWS CLI `config` fichier, ajoutez un profil nommé pour l'administrateur. Vous utilisez ce profil lors de l'exécution des AWS CLI commandes. Conformément au principe de sécurité du moindre privilège, nous vous recommandons de créer un rôle IAM distinct doté de privilèges spécifiques aux tâches effectuées. Pour plus d'informations sur les profils nommés, consultez la section Configuration et paramètres des fichiers d'identification dans le Guide de l'AWS Command Line Interface utilisateur.

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

3. Vérifiez la configuration à l'aide de la `help` commande suivante.

```
aws medical-imaging help
```

S'il AWS CLI est correctement configuré, une brève description d'AWS HealthImaging ainsi qu'une liste des commandes disponibles s'affichent.

HealthImaging Tutoriel AWS

Objectif

L'objectif de ce didacticiel est d'importer des fichiers DICOM P10 dans un magasin de HealthImaging données AWS et de les transformer en ensembles d'images composés de métadonnées et de cadres d'image (données en pixels). Après avoir importé les données DICOM, vous accédez aux ensembles d'images, aux métadonnées et aux cadres d'images en fonction de vos préférences d'accès à HealthImaging.

Prérequis

Toutes les procédures répertoriées dans Configuration sont nécessaires pour terminer ce didacticiel.

Étapes du didacticiel

1. Démarrer le travail d'importation

2. Obtenir les propriétés des tâches d'importation
3. Rechercher des ensembles d'images
4. Obtenir les propriétés du jeu d'images
5. Obtenir les métadonnées d'un ensemble d'images
6. Obtenir les données en pixels d'un ensemble d'images
7. Supprimer le magasin de données

Gestion des magasins de données avec AWS HealthImaging

Avec AWS HealthImaging, vous créez et gérez des magasins de données pour les ressources d'imagerie médicale. Les rubriques suivantes décrivent comment utiliser des HealthImaging actions pour créer, décrire, répertorier et supprimer des banques de données à l'aide des kits de AWS développement logiciel (SDK) et (SDK). AWS Management Console AWS CLI

> **Note**
>
> La dernière rubrique de ce chapitre traite de la compréhension des niveaux de stockage. Une fois que vous avez importé vos données d'imagerie médicale dans un magasin de données, celles-ci passent automatiquement d'un niveau de stockage à un autre en fonction du temps et de l'utilisation. Les niveaux de prix varient selon les niveaux de stockage. Il est donc important de comprendre le processus de transfert des niveaux et les HealthImaging ressources prises en compte à des fins de facturation.

Rubriques

Création d'un magasin de données

Utilisez cette `CreateDatastore` action pour créer un magasin de HealthImaging données AWS pour importer des fichiers DICOM P10. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `CreateDatastore` le manuel de référence des HealthImaging API AWS.

> **Important**
>
> Ne nommez pas les magasins de données contenant des informations de santé protégées (PHI), des informations personnelles identifiables (PII) ou d'autres informations confidentielles ou sensibles.

Pour créer un magasin de données

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page Créer un magasin de données de la HealthImaging console.
2. Sous Détails, dans Nom du magasin de données, entrez le nom de votre magasin de données.
3. Sous Chiffrement des données, choisissez une AWS KMS clé pour chiffrer vos ressources. Pour plus d'informations, consultez Protection des données dans AWS HealthImaging.
4. Sous Balises (facultatif), vous pouvez ajouter des balises à votre banque de données lorsque vous la créez. Pour plus d'informations, consultez Marquer une ressource.
5. Choisissez Créer un magasin de données.

AWS CLI et SDK

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, voir CreateDatastorela section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour créer un magasin de données

L'exemple de `create-datastore` code suivant crée un magasin de données portant le nom`my-datastore`.

```
aws medical-imaging create-datastore \
```

```
    --datastore-name "my-datastore"
```

Sortie :

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

Pour plus d'informations, consultez la section Création d'un magasin de données dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, voir CreateDatastorela section Référence des AWS CLI commandes.

Java

SDK pour 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 "";
   }
```

- Pour plus de détails sur l'API, voir CreateDatastorela section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, voir CreateDatastorela section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez CreateDatastorele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir les propriétés du magasin de données

Utilisez cette `GetDatastore` action pour récupérer les propriétés du magasin de HealthImaging données AWS. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `GetDatastore`le manuel de référence des HealthImaging API AWS.

Pour obtenir les propriétés du magasin de données

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre. Dans la section Détails, toutes les propriétés du magasin de données sont disponibles. Pour afficher les ensembles d'images, les importations et les balises associés, sélectionnez l'onglet approprié.

AWS CLI et SDK

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour obtenir les propriétés d'un magasin de données

L'exemple de `get-datastore` code suivant permet d'obtenir les propriétés d'un magasin de données.

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

Sortie :

```
{
    "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"
    }
}
```

Pour plus d'informations, consultez la section Obtenir les propriétés du magasin de données dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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"];
};
```

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetDatastorele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Lister les magasins de données

Utilisez cette `ListDatastores` action pour répertorier les magasins de données disponibles dans AWS HealthImaging. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `ListDatastores`le manuel de référence des HealthImaging API AWS.

Pour répertorier les magasins de données

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

- Ouvrez la page HealthImaging Stockages de données de la console.

  Tous les magasins de données sont répertoriés dans la section Magasins de données.

AWS CLI et SDK

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour répertorier les magasins de données

L'exemple de `list-datastores` code suivant répertorie les magasins de données disponibles.

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

Sortie :

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

Pour plus d'informations, consultez la section Répertorier les banques de données dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListDatastoresle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimer un magasin de données

Utilisez cette `DeleteDatastore` action pour supprimer un magasin de HealthImaging données AWS. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `DeleteDatastore`le manuel de référence des HealthImaging API AWS.

> **Note**
>
> Avant de pouvoir supprimer un magasin de données, vous devez d'abord supprimer tous les ensembles d'images qu'il contient. Pour plus d’informations, consultez Supprimer un ensemble d'images.

Pour supprimer un magasin de données

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.
3. Sélectionnez Delete (Supprimer).

   La page Supprimer le magasin de données s'ouvre.
4. Pour confirmer la suppression du magasin de données, entrez le nom du magasin de données dans le champ de saisie de texte.
5. Choisissez Supprimer le magasin de données.

AWS CLI et SDK

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, voir DeleteDatastorela section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour supprimer un magasin de données

L'exemple de `delete-datastore` code suivant supprime un magasin de données.

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

Sortie :

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

Pour plus d'informations, consultez la section Suppression d'un magasin de données dans le guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, voir DeleteDatastorela section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
   }
```

- Pour plus de détails sur l'API, voir DeleteDatastorela section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, voir DeleteDatastorela section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez DeleteDatastorele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Comprendre les niveaux de stockage

AWS HealthImaging utilise une hiérarchisation intelligente pour la gestion automatique du cycle de vie clinique. Cela se traduit par des performances et un prix attractifs, à la fois pour les données nouvelles ou actives et pour les données d'archivage à long terme, sans aucune friction. HealthImaging facture le stockage par Go et par mois en utilisant les niveaux suivants.

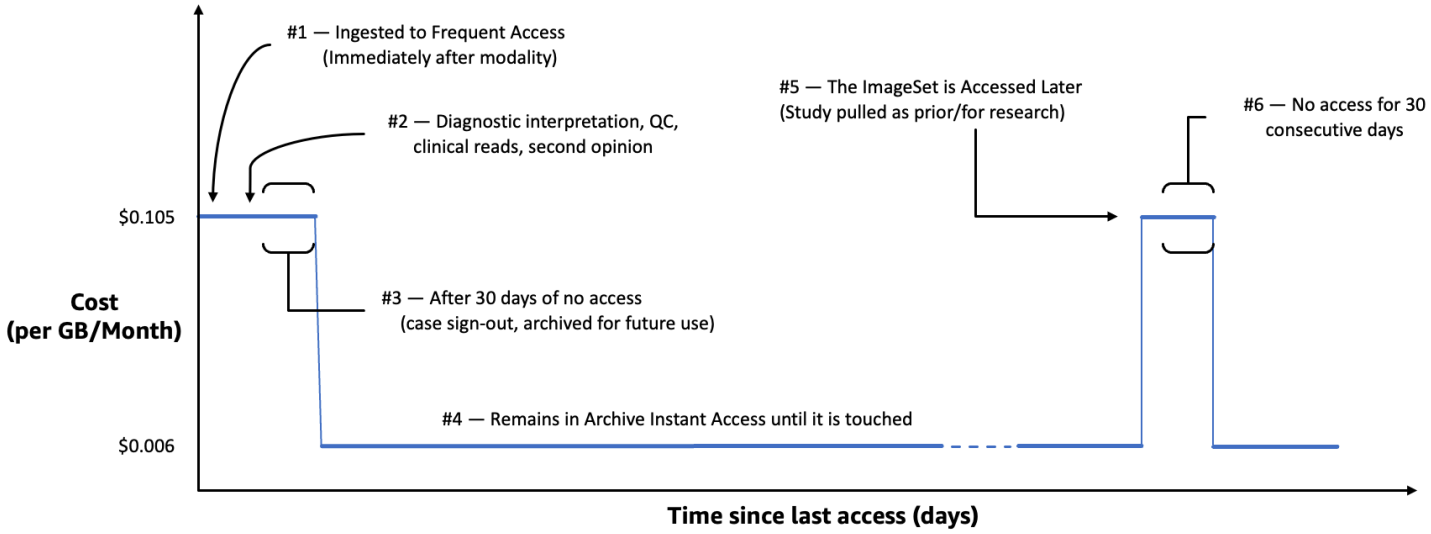
- Niveau d'accès fréquent : niveau pour les données fréquemment consultées.
- Niveau d'accès instantané aux archives : niveau pour les données archivées.

> **Note**
>
> Il n'y a pas de différence de performance entre les niveaux Frequent Access et Archive Instant Access. La hiérarchisation intelligente est appliquée à des actions d'API d'ensembles d'images spécifiques. La hiérarchisation intelligente ne reconnaît pas les actions d'API de stockage, d'importation et de balisage des données. Le déplacement entre les niveaux est automatique en fonction de l'utilisation de l'API et est expliqué dans la section suivante.

Comment fonctionne le transfert de niveaux ?

- Après l'importation, les ensembles d'images démarrent dans le niveau Frequent Access.
- Après 30 jours consécutifs sans contact, les ensembles d'images passent automatiquement au niveau Archive Instant Access.
- Les ensembles d'images du niveau Archive Instant Access ne repassent au niveau Accès fréquent qu'après avoir été touchés.

Le graphique suivant fournit une vue d'ensemble du processus de hiérarchisation HealthImaging intelligent.



Qu'est-ce qui est considéré comme une touche ?

Une touche est un accès à une API spécifique via AWS Management Console le ou AWS CLI AWS les SDK et se produit lorsque :

1. Un nouveau jeu d'images est créé (`StartDICOMImportJob`ou`CopyImageSet`)
2. Un ensemble d'images est mis à jour (`UpdateImageSetMetadata`ou`CopyImageSet`)
3. Les métadonnées ou le cadre d'image (données en pixels) associés à un ensemble d'images sont lus (`GetImageSetMetaData`ou`GetImageFrame`)

Les actions HealthImaging d'API suivantes permettent de toucher et de déplacer des ensembles d'images du niveau Archive Instant Access au niveau Frequent Access.

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

> **Note**
>
> Bien que les cadres d'image (données en pixels) ne puissent pas être supprimés à l'aide de `UpdateImageSetMetadata` cette action, ils sont toujours comptabilisés à des fins de facturation.

Les actions HealthImaging d'API suivantes n'entraînent pas de retouches. Par conséquent, ils ne déplacent pas les ensembles d'images du niveau Archive Instant Access vers le niveau Frequent Access.

- `CreateDatastore`
- `GetDatastore`
- `ListDatastores`
- `DeleteDatastore`
- `GetDICOMImportJob`
- `ListDICOMImportJobs`
- `SearchImageSets`
- `GetImageSet`
- `ListImageSetVersions`

- `DeleteImageSet`
- `TagResource`
- `ListTagsForResource`
- `UntagResource`

Importation de données d'imagerie avec AWS HealthImaging

L'importation est le processus qui consiste à déplacer vos données d'imagerie médicale d'un compartiment d'entrée Amazon S3 vers un magasin de HealthImaging données AWS. Lors de l'importation, AWS HealthImaging effectue une vérification des données en pixels avant de transformer vos fichiers DICOM P10 en ensembles d'images composés de métadonnées et de cadres d'image (données en pixels).

> **Tip**
>
> Une fois que vous vous serez familiarisé avec HealthImaging, nous vous encourageons à vous y rendre pour démarrer la mise en œuvre HealthImaging Exemples de projets AWS à l'aide de nos projets d'importation et de visualisation.

Les rubriques suivantes décrivent comment importer vos données d'imagerie médicale dans un magasin de données à l'aide HealthImaging des kits de AWS développement logiciel (SDK) et (SDK). AWS Management Console AWS CLI

Rubriques

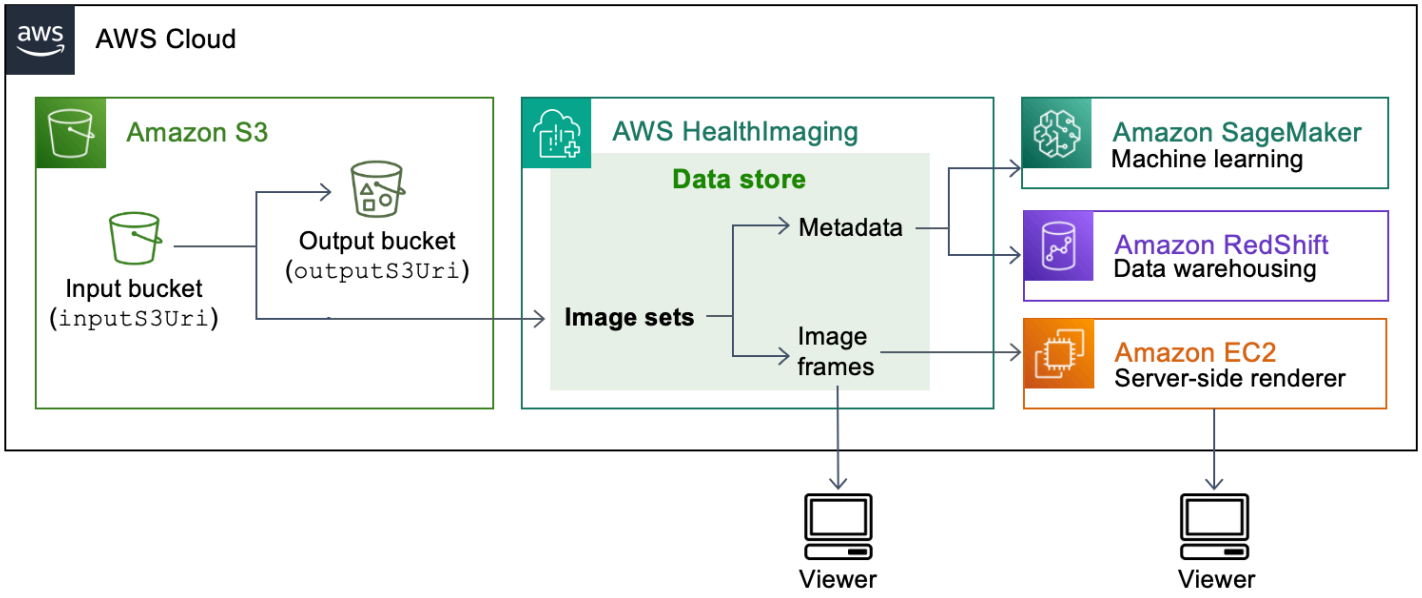
- Comprendre les tâches d'importation
- Démarrage d'une tâche d'importation
- Obtenir les propriétés des tâches d'importation
- Lister les tâches d'importation

Comprendre les tâches d'importation

Après avoir créé un magasin de données dans AWS HealthImaging, vous devez importer vos données d'imagerie médicale depuis votre compartiment d'entrée Amazon S3 vers votre magasin de données pour créer des ensembles d'images. Vous pouvez utiliser les AWS Management Console AWS kits SDK pour démarrer, décrire et répertorier les tâches d'importation. AWS CLI

Le schéma suivant donne un aperçu de la manière dont HealthImaging les données DICOM sont importées dans un magasin de données et les transforment en ensembles d'images. Les résultats

du traitement des tâches d'importation sont stockés dans le compartiment de sortie Amazon S3 (`outputS3Uri`) et les ensembles d'images sont stockés dans le magasin de HealthImaging données AWS.



Gardez les points suivants à l'esprit lorsque vous importez vos fichiers d'imagerie médicale depuis Amazon S3 vers un magasin de HealthImaging données AWS :

- Des classes SOP et des syntaxes de transfert spécifiques sont prises en charge pour les tâches d'importation. Pour plus d'informations, consultez Support DICOM.
- Des contraintes de longueur s'appliquent à des éléments DICOM spécifiques lors de l'importation. Pour garantir le succès de l'importation, vérifiez que vos données d'imagerie médicale ne dépassent pas les limites de longueur. Pour plus d'informations, consultez Contraintes relatives aux éléments DICOM.
- Un contrôle de vérification des données en pixels est effectué au début des tâches d'importation. Pour plus d'informations, consultez Vérification des données des pixels.
- Des points de terminaison, des quotas et des limites de limitation sont associés HealthImaging aux actions d'importation. Pour plus d'informations, consultez Points de terminaison et quotas et Limites d'étranglement.
- Pour chaque tâche d'importation, les résultats du traitement sont stockés sur `outputS3Uri` place. Les résultats du traitement sont organisés sous forme de `job-output-manifest.json` fichiers `SUCCESS` et de `FAILURE` dossiers.

> **Note**
>
> Vous pouvez inclure jusqu'à 10 000 dossiers imbriqués pour une seule tâche d'importation.

- Le `job-output-manifest.json` fichier contient des `jobSummary` résultats et des informations supplémentaires sur les données traitées. L'exemple suivant montre la sortie d'un `job-output-manifest.json` fichier.

```
{
    "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": "123456123456123456789017891789012",
                "numberOfMatchedSOPInstances": 1
            }
        ]
    }
}
```

- Le SUCCESS dossier contient le `success.ndjson` fichier contenant les résultats de tous les fichiers d'imagerie importés avec succès. L'exemple suivant montre la sortie d'un `success.ndjson` fichier.

```
{"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"}}
```

- Le FAILURE dossier contient le `failure.ndjson` fichier contenant les résultats de tous les fichiers d'imagerie qui n'ont pas été correctement importés. L'exemple suivant montre la sortie d'un `failure.ndjson` fichier.

```
{"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"}}
```

- Les tâches d'importation sont conservées dans la liste des tâches pendant 90 jours, puis archivées.

Démarrage d'une tâche d'importation

Utilisez cette `StartDICOMImportJob` action pour démarrer un contrôle de vérification des données de pixels et une importation de données en masse dans un magasin de HealthImaging données AWS. La tâche d'importation importe des fichiers DICOM P10 situés dans le compartiment d'entrée Amazon S3 spécifié par le `inputS3Uri` paramètre. Les résultats du traitement des tâches d'importation sont stockés dans le compartiment de sortie Amazon S3 spécifié par le `outputS3Uri` paramètre.

> **Note**
>
> HealthImaging prend en charge les importations de données à partir de compartiments Amazon S3 situés dans d'autres régions prises en charge. Pour obtenir cette fonctionnalité, fournissez le `inputOwnerAccountId` paramètre lors du démarrage d'une tâche d'importation. Pour plus d'informations, consultez Importation entre comptes pour AWS HealthImaging.

Lors de l'importation, des contraintes de longueur sont appliquées à des éléments DICOM spécifiques. Pour plus d'informations, consultez Contraintes relatives aux éléments DICOM.

Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `StartDICOMImportJob`le manuel de référence des HealthImaging API AWS.

Pour démarrer une tâche d'importation

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.
3. Choisissez Importer des données DICOM.

   La page Importer des données DICOM s'ouvre.
4. Dans la section Détails, entrez les informations suivantes :
   - Nom (facultatif)
   - Importer l'emplacement de la source dans S3
   - ID de compte du propriétaire du compartiment source (facultatif)
   - Clé de chiffrement (facultatif)
   - Destination de sortie dans S3
5. Dans la section Accès au service, choisissez Utiliser un rôle de service existant et sélectionnez le rôle dans le menu Nom du rôle de service ou choisissez Créer et utiliser un nouveau rôle de service.
6. Choisissez Import (Importer).

AWS CLI et SDK

C++

SDK pour 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();
}
```

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans la référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour démarrer une tâche d'importation DICOM

L'exemple de `start-dicom-import-job` code suivant démarre une tâche d'importation 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"
```

Sortie :

```
{
```

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

Pour plus d'informations, consultez la section Démarrage d'une tâche d'importation dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans AWS CLI Command Reference.

Java

SDK pour 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 "";
   }
```

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans la référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans la référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez StartDICOM ImportJob dans le manuel de référence de l'API AWS SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir les propriétés des tâches d'importation

Utilisez cette `GetDICOMImportJob` action pour en savoir plus sur les propriétés HealthImaging des tâches d'importation AWS. Par exemple, après avoir démarré une tâche d'importation, vous pouvez courir `GetDICOMImportJob` pour connaître le statut de la tâche. Une fois le `jobStatus` retour en tant que`COMPLETED`, vous êtes prêt à accéder à vos ensembles d'images.

> **Note**
>
> Le `jobStatus` fait référence à l'exécution de la tâche d'importation. Par conséquent, une tâche d'importation peut renvoyer un `jobStatus` as `COMPLETED` même si des problèmes de validation sont découverts au cours du processus d'importation. Si un tel `jobStatus` est le cas`COMPLETED`, nous vous recommandons tout de même de consulter les manifestes de sortie écrits sur Amazon S3, car ils fournissent des informations sur le succès ou l'échec des importations d'objets P10 individuels.

Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `GetDICOMImportJob`le manuel de référence des HealthImaging API AWS.

Pour obtenir les propriétés des tâches d'importation

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre. L'onglet Ensembles d'images est sélectionné par défaut.

3. Choisissez l'onglet Importations.
4. Choisissez une tâche d'importation.

   La page des détails de la tâche d'importation s'ouvre et affiche les propriétés relatives à la tâche d'importation.

AWS CLI et kits de développement logiciel

C++

SDK pour 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;
}
```

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans la section AWS SDK for C++ API Reference.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour obtenir les propriétés d'une tâche d'importation DICOM

L'exemple de `get-dicom-import-job` code suivant permet d'obtenir les propriétés d'une tâche d'importation DICOM.

```
aws medical-imaging get-dicom-import-job \
    --datastore-id "1234567890123456789012345678901 2" \
    --job-id "09876543210987654321098765432109"
```

Sortie :

```
{
    "jobProperties": {
        "jobId": "09876543210987654321098765432109",
        "jobName": "my-job",
        "jobStatus": "COMPLETED",
        "datastoreId": "1234567890123456789012345678901 2",
        "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/1234567890123456789012345678901 2-
DicomImport-09876543210987654321098765432109/"
    }
}
```

Pour plus d'informations, consultez la section Obtenir les propriétés des tâches d'importation dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans AWS CLI Command Reference.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans la section AWS SDK for Java 2.x API Reference.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans la section AWS SDK for JavaScript API Reference.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans le manuel de référence de l'API AWS SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Lister les tâches d'importation

Utilisez cette `ListDICOMImportJobs` action pour répertorier les tâches d'importation créées pour un magasin de HealthImaging données spécifique. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `ListDICOMImportJobs`le manuel de référence des HealthImaging API AWS.

> **Note**
>
> Les tâches d'importation sont conservées dans la liste des tâches pendant 90 jours, puis archivées.

Pour répertorier les tâches d'importation

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

La page de détails du magasin de données s'ouvre. L'onglet Ensembles d'images est sélectionné par défaut.

3. Cliquez sur l'onglet Importations pour répertorier toutes les tâches d'importation associées.

AWS CLI et SDK

CLI

AWS CLI

Pour répertorier les tâches d'importation Dicom

L'exemple de `list-dicom-import-jobs` code suivant répertorie les tâches d'importation dicom.

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

Sortie :

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

Pour plus d'informations, consultez la section Répertorier les tâches d'importation dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, voir ListDicom ImportJobs dans AWS CLI Command Reference.

Java

SDK pour 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<>();
   }
```

- Pour plus de détails sur l'API, voir ListDicom ImportJobs dans la référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- Pour plus de détails sur l'API, voir ListDicom ImportJobs dans la référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListDICOM ImportJobs dans le manuel de référence de l'API AWS SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Accès aux ensembles d'images avec AWS HealthImaging

L'accès aux données d'imagerie médicale dans AWS implique HealthImaging généralement de rechercher un ensemble d'images à l'aide d'une clé unique et d'obtenir les métadonnées et les cadres d'image (données en pixels) associés.

> **Tip**
>
> Une fois que vous vous serez familiarisé avec AWS HealthImaging, nous vous encourageons à vous y rendre pour démarrer la mise en œuvre HealthImaging Exemples de projets AWS à l'aide de nos projets de visualisation.

Les rubriques suivantes expliquent ce que sont les ensembles d'images et comment utiliser les AWS Management Console AWS SDK pour les rechercher et obtenir leurs propriétés, métadonnées et cadres d'image associés. AWS CLI

Rubriques

Comprendre les ensembles d'images

Les ensembles d'images sont un AWS concept qui constitue la base d'AWS HealthImaging. Les ensembles d'images sont créés lorsque vous importez vos données DICOM. Il est donc nécessaire de bien les comprendre lorsque vous travaillez avec le service. HealthImaging

Les ensembles d'images ont été introduits pour les raisons suivantes :

- Support d'une grande variété de flux de travail d'imagerie médicale (cliniques et non cliniques) grâce à des API flexibles.
- Optimisez la sécurité des patients en regroupant uniquement les données connexes.

- Encouragez le nettoyage des données afin d'accroître la visibilité des incohérences. Pour plus d'informations, consultez Modification de séries d'images.

> **Important**
>
> L'utilisation clinique des données DICOM avant leur nettoyage peut être préjudiciable au patient.

Les menus suivants décrivent les ensembles d'images de manière plus détaillée et fournissent des exemples et des diagrammes pour vous aider à comprendre leur fonctionnalité et leur objectif. HealthImaging

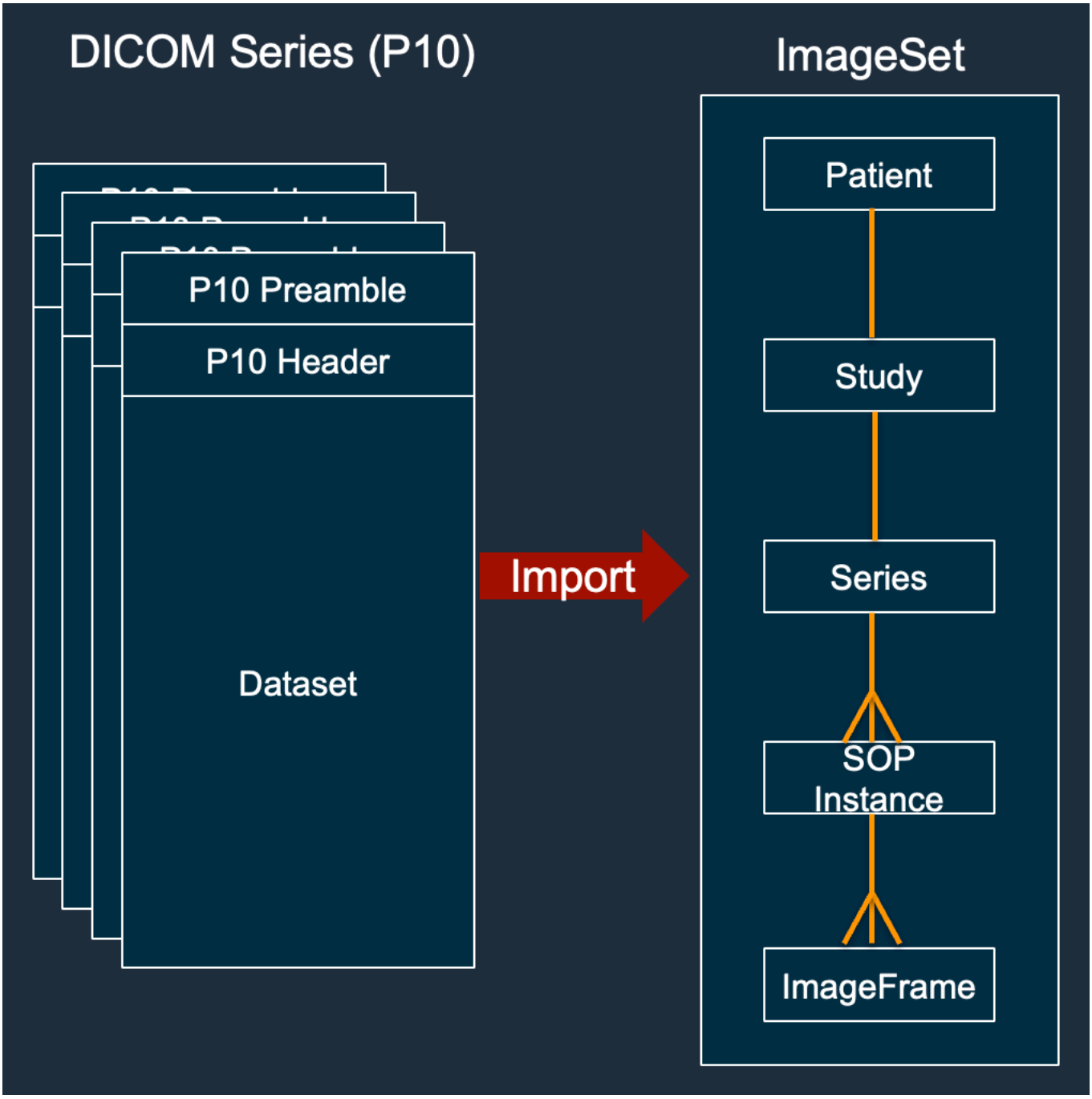
Qu'est-ce qu'une série d'images ?

Un ensemble d'images est un AWS concept qui définit un mécanisme de regroupement abstrait pour optimiser les données d'imagerie médicale associées. Lorsque vous importez vos données d'imagerie DICOM P10 dans un magasin de HealthImaging données AWS, elles sont transformées en ensembles d'images composés de métadonnées et de cadres d'image (données en pixels).

> **Note**
>
> Les métadonnées des ensembles d'images sont normalisées. En d'autres termes, un ensemble commun d'attributs et de valeurs correspond aux éléments au niveau du patient, de l'étude et de la série répertoriés dans le registre des éléments de données DICOM.
> Les images (données en pixels) sont codées au format JPEG 2000 à haut débit (HTJ2K) et doivent être décodées avant d'être visionnées.

Les ensembles d'images AWS étant des ressources, des noms de ressources Amazon (ARN) leur sont attribués. Ils peuvent être étiquetés avec jusqu'à 50 paires clé-valeur et bénéficier d'un contrôle d'accès basé sur les rôles (RBAC) et d'un contrôle d'accès basé sur les attributs (ABAC) via IAM. En outre, les ensembles d'images sont versionnés, de sorte que toutes les modifications sont préservées et que les versions antérieures sont accessibles.

L'importation de données DICOM P10 produit des ensembles d'images contenant des métadonnées DICOM et des cadres d'image pour une ou plusieurs instances de paire Service-Object (SOP) de la même série DICOM.



> **Note**
>
> Tâches d'importation DICOM :
>
> - Créez toujours de nouveaux ensembles d'images et ne mettez jamais à jour les ensembles d'images existants.

- Ne dédupliquez pas le stockage d'une instance SOP, car chaque importation de la même instance SOP utilise un espace de stockage supplémentaire.
- Peut créer plusieurs ensembles d'images pour une seule série DICOM. Par exemple, lorsqu'il existe une variante d'un attribut de métadonnées normalisé, telle qu'une `PatientName` non-concordance.

À quoi ressemblent les métadonnées des ensembles d'images ?

Utilisez cette `GetImageSetMetadata` action pour récupérer les métadonnées d'un ensemble d'images. Les métadonnées renvoyées sont compressées avec`gzip`. Vous devez donc les décompresser avant de les visualiser. Pour plus d'informations, consultez Obtenir les métadonnées d'un ensemble d'images.

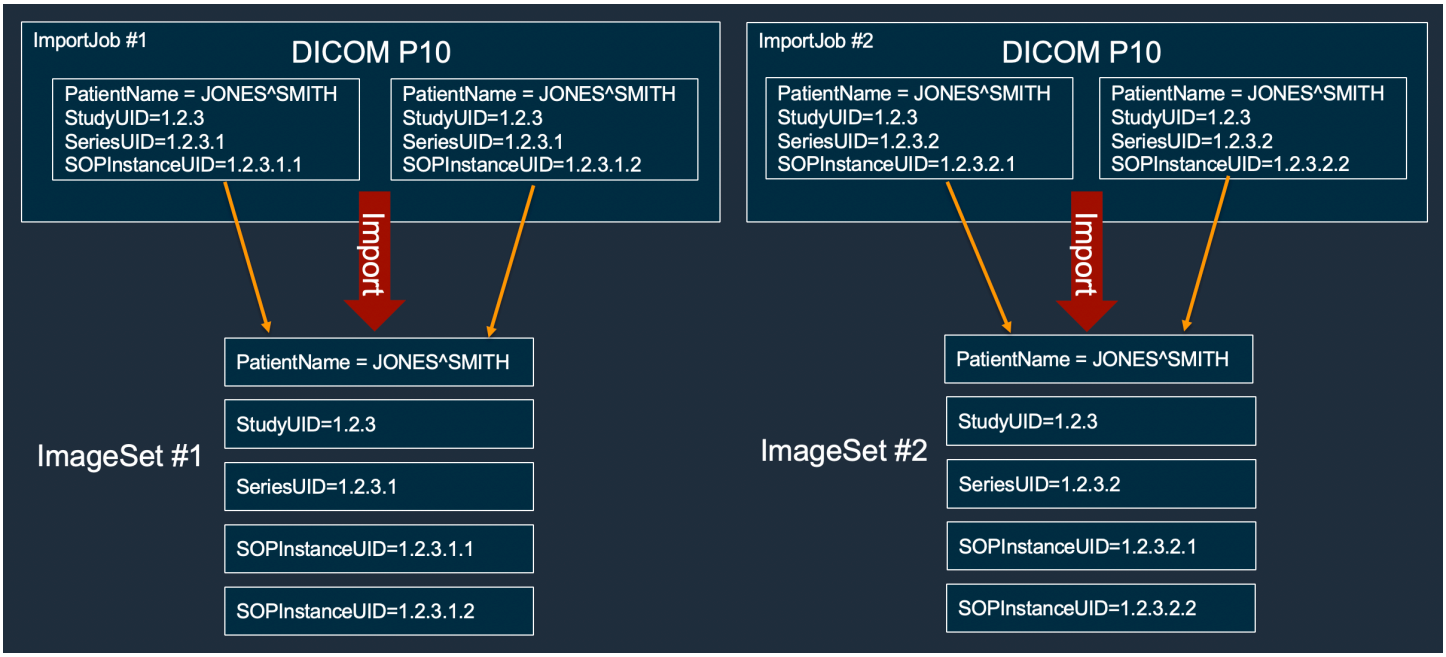
L'exemple suivant montre la structure des métadonnées des ensembles d'images au format JSON.

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

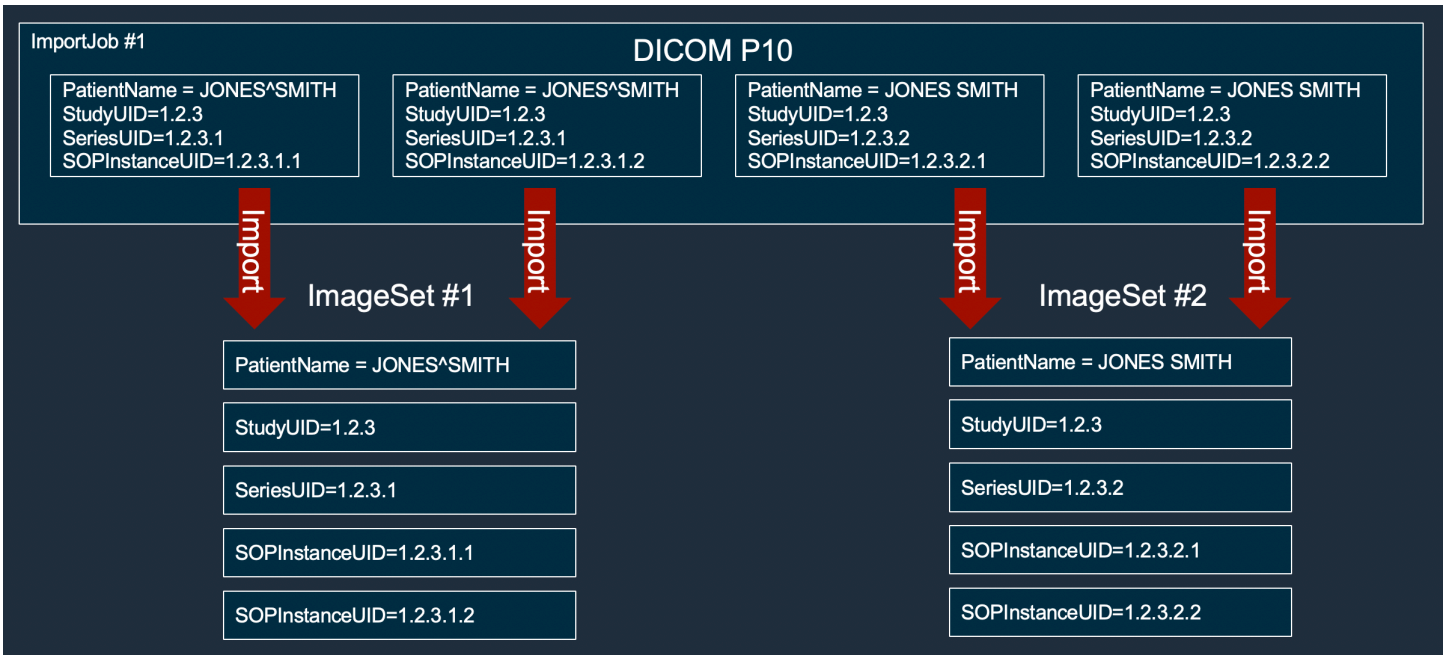
Exemple de création d'un jeu d'images : plusieurs tâches d'importation

L'exemple suivant montre comment plusieurs tâches d'importation créent toujours de nouveaux ensembles d'images et ne s'ajoutent jamais aux ensembles existants.



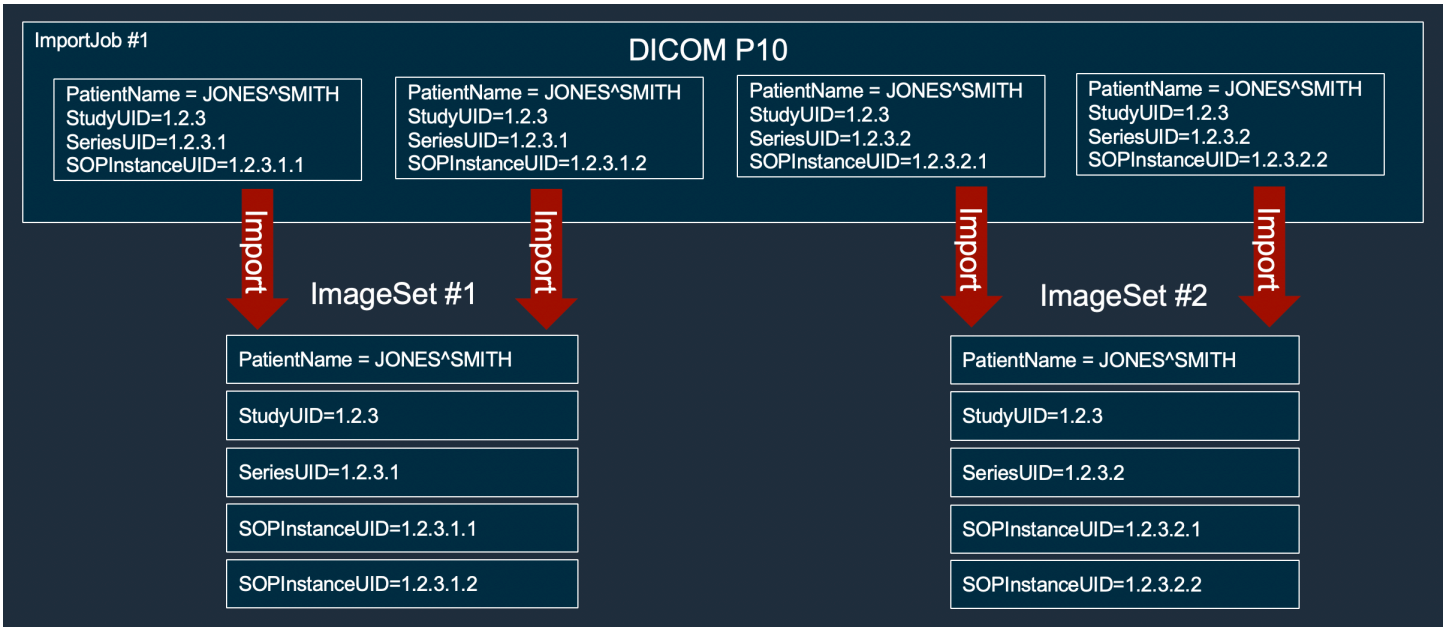
Exemple de création d'un jeu d'images : tâche d'importation unique avec deux variantes

L'exemple suivant montre une tâche d'importation unique créant deux ensembles d'images car les instances 1 et 2 portent des noms de patients différents de ceux des instances 3 et 4.



Exemple de création d'un jeu d'images : tâche d'importation unique avec optimisation

L'exemple suivant montre une seule tâche d'importation créant deux séries d'images pour améliorer le débit, même si les noms des patients correspondent.



Recherche de séries d'images

Utilisez cette `SearchImageSets` action pour exécuter des requêtes de recherche sur tous les ensembles d'images d'un magasin de `ACTIVE` HealthImaging données. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `SearchImageSets`le manuel de référence des HealthImaging API AWS.

> **Note**
>
> Gardez les points suivants à l'esprit lorsque vous recherchez des ensembles d'images.
>
> - `SearchImageSets`accepte un seul paramètre de requête de recherche et renvoie une réponse paginée de tous les ensembles d'images répondant aux critères correspondants. Toutes les requêtes de plage de dates doivent être saisies sous la forme`(lowerBound, upperBound)`.
> - Par défaut, `SearchImageSets` utilise le `updatedAt` champ pour trier par ordre décroissant du plus récent au plus ancien.

- Si vous avez créé votre banque de données avec une AWS KMS clé appartenant au client, vous devez mettre à jour votre politique en matière de AWS KMS clés avant d'interagir avec des ensembles d'images. Pour plus d'informations, consultez la section Création d'une clé gérée par le client.

Pour rechercher des ensembles d'images

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

> **Note**
>
> Les procédures suivantes indiquent comment rechercher des ensembles d'images à l'aide des filtres de `Updated at` propriétés `Series Instance UID` et.

Series Instance UID

Rechercher des ensembles d'images à l'aide du filtre de **`Series Instance UID`** propriétés

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre et l'onglet Ensembles d'images est sélectionné par défaut.
3. Choisissez le menu du filtre de propriétés, puis sélectionnez`Series Instance UID`.
4. Dans le champ Entrez la valeur à rechercher, entrez (collez) l'UID de l'instance de série qui vous intéresse.

   > **Note**
   >
   > Les valeurs de l'UID de l'instance de série doivent être identiques à celles répertoriées dans le registre des identificateurs uniques (UID) DICOM. Notez que les exigences incluent une série de chiffres séparés par au moins un point. Les périodes ne sont pas autorisées au début ou à la fin des UID des instances de série. Les

lettres et les espaces blancs ne sont pas autorisés. Soyez donc prudent lorsque vous copiez et collez des UID.

5. Choisissez le menu Plage de dates, sélectionnez une plage de dates pour l'UID de l'instance de série, puis choisissez Appliquer.
6. Choisissez Rechercher.

   Les UID des instances de série compris dans la plage de dates sélectionnée sont renvoyés par défaut dans l'ordre le plus récent.

Updated at

Rechercher des ensembles d'images à l'aide du filtre de **`Updated at`** propriétés

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre et l'onglet Ensembles d'images est sélectionné par défaut.
3. Choisissez le menu du filtre de propriétés, puis choisissez`Updated at`.
4. Choisissez le menu Plage de dates, sélectionnez une plage de dates définie pour les images, puis choisissez Appliquer.
5. Choisissez Rechercher.

   Les ensembles d'images compris dans la plage de dates sélectionnée sont renvoyés par défaut dans l'ordre le plus récent.

AWS CLI et SDK

C++

SDK pour C++

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
//! 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;
```

```
}
```

Cas d'utilisation #1 : opérateur 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;
           }
       }
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
        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;
            }
        }
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
        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;
            }
        }
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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;
            }
        }
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Exemple 1 : Pour rechercher des ensembles d'images avec un opérateur EQUAL

L'exemple de `search-image-sets` code suivant utilise l'opérateur EQUAL pour rechercher des ensembles d'images en fonction d'une valeur spécifique.

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

Contenu de `search-criteria.json`

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

Sortie :

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

Exemple 2 : Pour rechercher des ensembles d'images avec un opérateur BETWEEN à l'aide de DICOM StudyDate et DICOM StudyTime

L'exemple de `search-image-sets` code suivant recherche des ensembles d'images contenant des études DICOM générées entre le 1er janvier 1990 (00h00) et le 1er janvier 2023 (00h00).

Remarque : le format DICOM StudyTime est facultatif. S'il n'est pas présent, 00 h 00 (début de journée) est la valeur horaire pour les dates fournies pour le filtrage.

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

Contenu de `search-criteria.json`

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

Sortie :

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

Exemple 3 : Pour rechercher des ensembles d'images avec un opérateur BETWEEN à l'aide de CreatedAt (les études temporelles étaient précédemment conservées)

L'exemple de `search-image-sets` code suivant recherche des ensembles d'images contenant des études DICOM persistantes HealthImaging entre les plages horaires du fuseau horaire UTC.

Remarque : Indiquez CreatedAt dans le format d'exemple (« 1985-04-12T 23:20:50,52 Z »).

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

Contenu de `search-criteria.json`

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

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

Sortie :

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

Exemple 4 : Pour rechercher des ensembles d'images avec un opérateur EQUAL sur DICOM SeriesInstance UID et BETWEEN sur UpdateDat et trier les réponses dans l'ordre ASC sur le champ UpdatedAt

L'exemple de `search-image-sets` code suivant recherche des ensembles d'images avec un opérateur EQUAL sur DICOM SeriesInstance UID et BETWEEN sur UpdateDat et trie la réponse dans l'ordre ASC sur le champ UpdatedAt.

Remarque : Fournissez UpdateDat dans le format d'exemple (« 1985-04-12T 23:20:50,52 Z »).

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

Contenu de `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"
    }
}
```

Sortie :

```
{
    "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"
```

```
    }]
}
```

Pour plus d'informations, consultez la section Recherche de séries d'images dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
    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;
    }
```

Cas d'utilisation #1 : opérateur 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();
        }
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
        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();
        }
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
        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();
        }
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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();
        }
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
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;
};
```

Cas d'utilisation #1 : opérateur EQUAL.

```
    const datastoreId = "12345678901234567890123456789012";

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

```
            },
        ]
    };

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

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
const datastoreId = "1234567890123456789012345678901 2";

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);
}
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
    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);
    }
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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);
    }
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
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
```

Cas d'utilisation #1 : opérateur 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}")
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
        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}"
        )
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
        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}"
        )
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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}")
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez SearchImageSetsle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir les propriétés d'un ensemble d'images

Utilisez l'`GetImageSet`action pour renvoyer les propriétés d'une image donnée définie dans HealthImaging. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `GetImageSet`le manuel de référence des HealthImaging API AWS.

> **Note**
>
> Par défaut, AWS HealthImaging renvoie les propriétés de la dernière version d'un ensemble d'images. Pour afficher les propriétés d'une ancienne version d'un ensemble d'images, veuillez les `versionId` joindre à votre demande.

Pour obtenir les propriétés des ensembles d'images

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre et l'onglet Ensembles d'images est sélectionné par défaut.
3. Choisissez un ensemble d'images.

   La page de détails du jeu d'images s'ouvre et affiche les propriétés du jeu d'images.

AWS CLI et SDK

CLI

AWS CLI

Pour obtenir les propriétés des ensembles d'images

L'exemple de `get-image-set` code suivant permet d'obtenir les propriétés d'un ensemble d'images.

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

Sortie :

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

Pour plus d'informations, consultez la section Obtenir les propriétés d'un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetImageSetle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir les métadonnées d'un ensemble d'images

Utilisez l'`GetImageSetMetadata`action pour récupérer les métadonnées d'une image donnée définie dans HealthImaging. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `GetImageSetMetadata`le manuel de référence des HealthImaging API AWS.

> **Note**
>
> Par défaut, HealthImaging renvoie les attributs de métadonnées pour la dernière version d'un ensemble d'images. Pour consulter les métadonnées d'une ancienne version d'un ensemble d'images, `versionId` joignez-les à votre demande.
> Les métadonnées des ensembles d'images sont compressées `gzip` et renvoyées sous forme d'objet JSON. Par conséquent, vous devez décompresser l'objet JSON avant de visualiser les métadonnées normalisées. Pour plus d'informations, consultez Normalisation des métadonnées.

Pour obtenir les métadonnées du jeu d'images

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre et l'onglet Ensembles d'images est sélectionné par défaut.
3. Choisissez un ensemble d'images.

   La page de détails du jeu d'images s'ouvre et les métadonnées du jeu d'images s'affichent dans la section Visionneuse de métadonnées du jeu d'images.

AWS CLI et SDK

C++

SDK pour C++

Fonction utilitaire pour obtenir les métadonnées d'un ensemble d'images.

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

Obtenez les métadonnées des ensembles d'images sans version.

```
        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;
        }
```

Obtenez les métadonnées des ensembles d'images avec la version.

```
        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;
        }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Exemple 1 : Pour obtenir les métadonnées d'un ensemble d'images sans version

L'exemple de `get-image-set-metadata` code suivant permet d'obtenir les métadonnées d'un ensemble d'images sans spécifier de version.

Remarque : `outfile` est un paramètre obligatoire

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

Les métadonnées renvoyées sont compressées avec gzip et stockées dans le fichier studymetadata.json.gz. Pour visualiser le contenu de l'objet JSON renvoyé, vous devez d'abord le décompresser.

Sortie :

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

Exemple 2 : Pour obtenir les métadonnées d'un ensemble d'images avec la version

L'exemple de `get-image-set-metadata` code suivant permet d'obtenir les métadonnées d'un ensemble d'images avec une version spécifiée.

Remarque : `outfile` est un paramètre obligatoire

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

Les métadonnées renvoyées sont compressées avec gzip et stockées dans le fichier studymetadata.json.gz. Pour visualiser le contenu de l'objet JSON renvoyé, vous devez d'abord le décompresser.

Sortie :

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

Pour plus d'informations, consultez la section Obtenir les métadonnées d'un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
```

```
    }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Fonction utilitaire pour obtenir les métadonnées d'un ensemble d'images.

```
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;
};
```

Obtenez les métadonnées des ensembles d'images sans version.

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

Obtenez les métadonnées des ensembles d'images avec la version.

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
```

```
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Fonction utilitaire pour obtenir les métadonnées d'un ensemble d'images.

```
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
```

Obtenez les métadonnées des ensembles d'images sans version.

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

Obtenez les métadonnées des ensembles d'images avec la version.

```
                image_set_metadata =
self.health_imaging_client.get_image_set_metadata(
                    imageSetId=image_set_id,
```

```
                datastoreId=datastore_id,
                versionId=version_id,
            )
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetImageSetMetadatale AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir les données en pixels d'un ensemble d'images

Une trame d'image correspond aux données de pixels qui existent dans un ensemble d'images pour constituer une image médicale en 2D. Utilisez cette `GetImageFrame` action pour récupérer un cadre d'image codé en HTJ2K pour une image donnée définie dans. HealthImaging Les menus suivants fournissent des exemples de code pour les AWS SDK AWS CLI et. Pour plus d'informations, consultez `GetImageFrame`le manuel de référence des HealthImaging API AWS.

> **Note**
>
> Lors de l'importation, AWS HealthImaging code toutes les images au format HTJ2K sans perte. Par conséquent, elles doivent être décodées avant d'être visualisées dans une visionneuse d'images. Pour plus d’informations, consultez bibliothèques de décodage HTJ2K.

Pour obtenir un cadre d'image

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

> **Note**
>
> Les images doivent être accessibles et décodées par programme, car aucune visionneuse d'images n'est disponible dans le. AWS Management Console
> Pour plus d'informations sur le décodage et l'affichage de cadres d'image, consultezbibliothèques de décodage HTJ2K.

AWS CLI et SDK

C++

SDK pour 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();
}
```

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour obtenir les données relatives aux pixels définis par image

L'exemple de `get-image-frame` code suivant permet d'obtenir un cadre d'image.

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

```
    imageframe.jph
```

Remarque : Cet exemple de code n'inclut pas de sortie car l' GetImageFrame action renvoie un flux de données de pixels vers le fichier imageframe.jph. Pour plus d'informations sur le décodage et l'affichage de trames d'image, consultez la section Bibliothèques de décodage HTJ2K.

Pour plus d'informations, consultez la section Obtenir des données en pixels d'un ensemble d'images dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
                }
        }
```

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetImageFramele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir une instance DICOM

> **Note**
>
> L' HealthImaging `GetDICOMInstance`API est conçue conformément à la norme DICOMWeb Retrieve (WADO-RS) pour l'imagerie médicale basée sur le Web. Comme il `GetDICOMInstance` s'agit d'une représentation d'un service DICOMWeb, il n'est pas proposé par le biais AWS CLI de AWS SDK.

Utilisez cette `GetDICOMInstance` action pour récupérer une instance DICOM (`.dcm`fichier) d'un magasin de HealthImaging données en spécifiant les UID de série, d'étude et d'instance associés à la ressource. Vous pouvez spécifier le jeu d'images à partir duquel une ressource d'instance doit être extraite en fournissant l'ID du jeu d'images en tant que paramètre de requête. En outre, vous pouvez choisir la syntaxe de transfert pour compresser les données DICOM, en prenant en charge le format non compressé (ELE) ou le JPEG 2000 à haut débit (HTJ2K). Vous pouvez `GetDICOMInstance` ainsi interagir avec des systèmes qui utilisent les fichiers binaires DICOM Part 10 tout en tirant parti des interfaces natives HealthImaging du cloud.

Pour obtenir une instance DICOM (**`.dcm`**fichier)

1. Collectez HealthImaging `datastoreId` et `imageSetId` paramétrez les valeurs.
2. Utilisez l'`GetImageSetMetadata`action avec les valeurs `datastoreId` des `imageSetId` paramètres et pour récupérer les valeurs de métadonnées associées pour `studyInstanceUID`, et`seriesInstanceUID`, et`sopInstanceUID`. Pour plus d'informations, consultez Obtenir les métadonnées d'un ensemble d'images.
3. Créez une URL pour la demande en utilisant les valeurs pour `datastoreId`,`studyInstanceUID`,`seriesInstanceUID`,`sopInstanceUID`, et`imageSetId`. L'URL est de la forme suivante :

   ```
   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. Préparez et envoyez votre demande. `GetDICOMInstance`utilise une requête HTTP GET avec le protocole de AWS signature Signature Version 4. L'exemple de code suivant utilise

l'outil de ligne de `curl` commande pour obtenir une instance DICOM (`.dcm`fichier) à partir de 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 est facultatif et est défini par défaut sur Explicit VR Little Endian s'il n'est pas inclus. Les syntaxes de transfert prises en charge incluent :
>
> - Explicit VR Little Endian (ELE) - `1.2.840.10008.1.2.1` (par défaut)
> - JPEG 2000 à haut débit avec options de compression d'image RPCL (sans perte uniquement) - `1.2.840.10008.1.2.4.202`
>
> Pour plus d'informations, voir bibliothèques de décodage HTJ2K pour AWS HealthImaging.

Modifier des ensembles d'images avec AWS HealthImaging

Les tâches d'importation DICOM nécessitent généralement que vous modifiiez vos ensembles d'images pour les raisons suivantes :

- Sécurité des patients
- Cohérence des données
- Réduisez les coûts de stockage

HealthImaging fournit plusieurs API pour simplifier le processus de modification des ensembles d'images. Les rubriques suivantes décrivent comment modifier des ensembles d'images à l'aide AWS CLI des AWS SDK et.

Rubriques

Liste des versions des ensembles d'images

Utilisez cette `ListImageSetVersions` action pour répertorier l'historique des versions d'une image définie dans HealthImaging. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `ListImageSetVersions`le manuel de référence des HealthImaging API AWS.

> **Note**
>
> AWS HealthImaging enregistre chaque modification apportée à un ensemble d'images. La mise à jour des métadonnées du jeu d'images crée une nouvelle version dans l'historique du jeu d'images. Pour plus d'informations, consultez Mise à jour des métadonnées du jeu d'images.

Pour répertorier les versions d'un ensemble d'images

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre et l'onglet Ensembles d'images est sélectionné par défaut.
3. Choisissez un ensemble d'images.

   La page de détails du jeu d'images s'ouvre.

   La version du jeu d'images s'affiche dans la section Détails du jeu d'images.

AWS CLI et SDK

CLI

AWS CLI

Pour répertorier les versions d'ensembles d'images

L'exemple de `list-image-set-versions` code suivant répertorie l'historique des versions d'un ensemble d'images.

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

Sortie :

```
{
    "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
        }
    ]
}
```

Pour plus d'informations, consultez la section Répertorier les versions des ensembles d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListImageSetVersionsle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

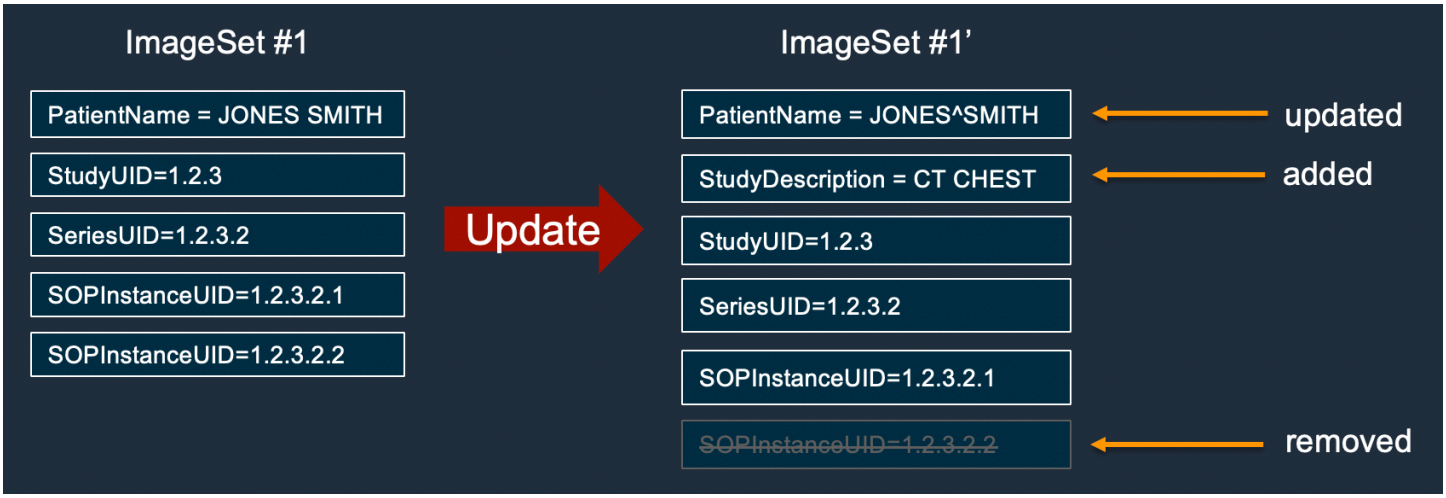
Mise à jour des métadonnées du jeu d'images

Utilisez cette `UpdateImageSetMetadata` action pour mettre à jour les métadonnées des ensembles d'images dans AWS HealthImaging. Vous pouvez utiliser ce processus asynchrone pour ajouter, mettre à jour et supprimer des attributs de métadonnées d'ensembles d'images, qui sont des manifestations des éléments de normalisation DICOM créés lors de l'importation. À l'aide de `UpdateImageSetMetadata` cette action, vous pouvez également supprimer des instances de série et de SOP pour synchroniser les ensembles d'images avec les systèmes externes et pour anonymiser les métadonnées des ensembles d'images. Pour plus d'informations, consultez `UpdateImageSetMetadata`le manuel de référence des HealthImaging API AWS.

Présentation du code **`UpdateImageSetMetadata`**

> **Note**
>
> Les importations DICOM réelles nécessitent la mise à jour, l'ajout et la suppression d'attributs dans les métadonnées du jeu d'images. Tenez compte des points suivants lorsque vous mettez à jour les métadonnées d'un ensemble d'images :
>
> - La mise à jour des métadonnées du jeu d'images crée une nouvelle version dans l'historique du jeu d'images. Pour plus d'informations, consultez Liste des versions des ensembles d'images.
> - La mise à jour des métadonnées des ensembles d'images est un processus asynchrone. Par conséquent, `imageSetState`des éléments de `imageSetWorkflowStatus`réponse sont disponibles pour fournir l'état et le statut respectifs d'un ensemble d'images verrouillé. Vous ne pouvez pas effectuer d'autres opérations d'écriture sur un jeu d'images verrouillé.
> - Les contraintes relatives aux éléments DICOM sont appliquées aux mises à jour des métadonnées. Pour plus d'informations, consultez Contraintes relatives aux métadonnées DICOM.
> - Si une action de mise à jour des métadonnées d'un ensemble d'images échoue, appelez et vérifiez l'élément de `message`réponse.

Le schéma suivant représente les métadonnées des ensembles d'images mises à jour dans HealthImaging.



Pour mettre à jour les métadonnées d'un ensemble d'images

Choisissez un onglet en fonction de vos préférences d'accès à AWS HealthImaging.

AWS CLI et SDK

CLI

AWS CLI

Pour insérer ou mettre à jour un attribut dans les métadonnées d'un ensemble d'images

L'exemple de `update-image-set-metadata` code suivant insère ou met à jour un attribut dans les métadonnées du jeu d'images.

```
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
```

Contenu de `metadata-updates.json`

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

Remarque : `updatableAttributes` est une chaîne JSON codée en Base64. Voici la chaîne JSON non codée.

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

Sortie :

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

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

Pour supprimer un attribut des métadonnées d'un ensemble d'images

L'exemple de `update-image-set-metadata` code suivant supprime un attribut des métadonnées du jeu d'images.

```
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
```

Contenu de `metadata-updates.json`

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

Remarque : `removableAttributes` est une chaîne JSON codée en Base64. Voici la chaîne JSON non codée. La clé et la valeur doivent correspondre à l'attribut à supprimer.

{» SchemaVersion « :1.1, "Étude » : {"DICOM » : {» StudyDescription « 0.0.CHEST"}}}

Sortie :

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

Pour supprimer une instance des métadonnées du jeu d'images

L'exemple de `update-image-set-metadata` code suivant supprime une instance des métadonnées du jeu d'images.

```
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
```

Contenu de `metadata-updates.json`

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

Remarque : `removableAttributes` est une chaîne JSON codée en Base64. Voici la chaîne JSON non codée.

{« 1.1.1.1.1.1.12345.123456789012.123.12345678901234.1" : {" Instances » : {"1.1.1.1.12345.123456789012.123.12345678901234.1" : {}}}}

Sortie :

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

Pour plus d'informations, consultez la section Mise à jour des métadonnées des ensembles d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
   }
```

Cas d'utilisation #1 : Insérer ou mettre à jour un attribut.

```
           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);
```

Cas d'utilisation #2 : Supprimer un attribut.

```
            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);
```

Cas d'utilisation #3 : Supprimer une instance.

```
            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);
```

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

Cas d'utilisation #1 : Insérer ou mettre à jour un attribut.

```
        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);
```

Cas d'utilisation #2 : Supprimer un attribut.

```
        // 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);
```

Cas d'utilisation #3 : Supprimer une instance.

```
        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);
```

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

Cas d'utilisation #1 : Insérer ou mettre à jour un attribut.

```
            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
            )
```

Cas d'utilisation #2 : Supprimer un attribut.

```
            # 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
            )
```

Cas d'utilisation #3 : Supprimer une instance.

```
            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
            )
```

- Pour plus de détails sur l'API, consultez UpdateImageSetMetadatale AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Copier un ensemble d'images

Utilisez cette `CopyImageSet` action pour copier une image définie dans HealthImaging. Ce processus asynchrone permet de copier le contenu d'une série d'images dans une série d'images nouvelle ou existante. Vous pouvez copier dans une nouvelle image pour diviser un ensemble d'images, ainsi que pour créer une copie séparée. Vous pouvez également copier dans un jeu d'images existant pour fusionner deux ensembles d'images. Pour plus d'informations, consultez `CopyImageSet`le manuel de référence des HealthImaging API AWS.

Présentation du code **CopyImageSet**

> **Note**
>
> Tenez compte des points suivants lorsque vous copiez un ensemble d'images :
>
> - La copie d'un jeu d'images crée une nouvelle version dans l'historique du jeu d'images. Pour plus d'informations, consultez Liste des versions des ensembles d'images.
> - La copie d'un ensemble d'images est un processus asynchrone. Par conséquent, les éléments de réponse state (`imageSetStateimageSetWorkflowStatus`) et status () sont disponibles pour vous indiquer quelle opération est en cours sur un ensemble d'images verrouillé. Les autres opérations d'écriture ne peuvent pas être effectuées sur un ensemble d'images verrouillé.
> - `CopyImageSet`nécessite des UID d'instance SOP uniques pour réussir. Par conséquent, vous devez sélectionner l'instance SOP appropriée en la supprimant du jeu d'images indésirable.
> - Si une action de copie d'un ensemble d'images échoue, appelez `GetImageSet` et vérifiez la `message`propriété. Pour plus d'informations, consultez Obtenir les propriétés d'un ensemble d'images.
> - Les importations DICOM réelles peuvent donner lieu à plusieurs ensembles d'images par série DICOM. Tenez compte des points suivants lorsque vous utilisez l'`CopyImageSet`action :
>   - Copie les instances d'un ensemble d'images vers un autre
>   - La copie nécessite que les deux ensembles d'images aient des métadonnées cohérentes

Pour copier un ensemble d'images

Choisissez un onglet en fonction de vos préférences d'accès à AWS HealthImaging.

AWS CLI et kits de développement logiciel

CLI

AWS CLI

Exemple 1 : pour copier une série d'images sans destination.

L'exemple de `copy-image-set` code suivant crée une copie dupliquée d'un ensemble d'images sans destination.

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

Sortie :

```
{
    "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"
}
```

Exemple 2 : pour copier une série d'images avec une destination.

L'exemple de `copy-image-set` code suivant crée une copie dupliquée d'un ensemble d'images avec une destination.

```
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"} }'
```

Sortie :

```
{
    "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"
}
```

Pour plus d'informations, consultez la section Copier un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS CLI commandes.

Java

SDK pour 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 "";
    }
```

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Fonction utilitaire pour copier un ensemble d'images.

```
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: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

Copiez un ensemble d'images sans destination.

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

Copiez un ensemble d'images avec une destination.

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

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Fonction utilitaire pour copier un ensemble d'images.

```
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"]
```

Copiez un ensemble d'images sans destination.

```
            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,
            )
```

Copiez un ensemble d'images avec une destination.

```
            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,
            )
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez CopyImageSetle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimer un ensemble d'images

Utilisez cette `DeleteImageSet` action pour supprimer une image définie dans HealthImaging. Les menus suivants fournissent une procédure AWS Management Console et des exemples de code pour AWS CLI les AWS SDK. Pour plus d'informations, consultez `DeleteImageSet`le manuel de référence des HealthImaging API AWS.

Pour supprimer un ensemble d'images

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre et l'onglet Ensembles d'images est sélectionné par défaut.
3. Choisissez un ensemble d'images, puis cliquez sur Supprimer.

   Le modal Supprimer le jeu d'images s'ouvre.
4. Indiquez l'ID du jeu d'images et choisissez Supprimer le jeu d'images.

AWS CLI et SDK

C++

SDK pour 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();
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour supprimer un ensemble d'images

L'exemple de `delete-image-set` code suivant supprime un ensemble d'images.

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

Sortie :

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

Pour plus d'informations, consultez la section Suppression d'un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez DeleteImageSetle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Marquage des ressources avec AWS HealthImaging

Vous pouvez attribuer des métadonnées aux HealthImaging ressources AWS (magasins de données et ensembles d'images) sous forme de balises. Chaque balise est une étiquette composée d'une clé définie par l'utilisateur et d'une valeur. Les balises vous aident à gérer, identifier, organiser, rechercher et filtrer les ressources.

> **Important**
>
> Ne stockez pas d'informations de santé protégées (PHI), d'informations personnelles identifiables (PII) ou d'autres informations confidentielles ou sensibles dans des balises. Les étiquettes ne sont pas destinées à être utilisées pour des données privées ou sensibles.

Les rubriques suivantes décrivent comment utiliser les opérations de HealthImaging balisage à l'aide des kits de développement logiciel (SDK) et AWS (SDK). AWS Management Console AWS CLI Pour plus d'informations, consultez la section Marquage de vos AWS ressources dans le Références générales AWS guide.

Rubriques

- Marquer une ressource
- Répertorier les tags d'une ressource
- Débalisage d'une ressource

Marquer une ressource

Utilisez cette `TagResource`action pour baliser une ressource dans AWS HealthImaging. Les exemples de code suivants décrivent comment utiliser l'`TagResource`action avec les AWS SDK AWS Management Console AWS CLI,, et. Pour plus d'informations, consultez la section Marquage de vos AWS ressources dans le Références générales AWS guide.

Pour étiqueter une ressource (magasin de données)

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre.
3. Cliquez sur l'onglet Détails.
4. Dans la section Balises, choisissez Gérer les balises.

   La page Gérer les balises s'ouvre.
5. Sélectionnez Ajouter une nouvelle balise.
6. Entrez une clé et une valeur (facultatif).
7. Sélectionnez Enregistrer les modifications.

AWS CLI et kits de développement logiciel

CLI

AWS CLI

Exemple 1 : pour étiqueter un magasin de données

Les exemples de `tag-resource` code suivants balisent un magasin de données.

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

Cette commande ne produit aucun résultat.

Exemple 2 : pour baliser un ensemble d'images

Les exemples de `tag-resource` code suivants balisent un ensemble d'images.

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

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

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Marquage des ressources avec AWS HealthImaging dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
   }
```

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez TagResourcele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertorier les tags d'une ressource

Utilisez cette `ListTagsForResource`action pour répertorier les balises d'une ressource dans AWS HealthImaging. Les exemples de code suivants décrivent comment utiliser l'`ListTagsForResource`action avec les AWS SDK AWS Management Console AWS CLI,, et. Pour plus d'informations, consultez la section Marquage de vos AWS ressources dans le Références générales AWS guide.

Pour répertorier les balises d'une ressource (magasin de données)

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre.
3. Cliquez sur l'onglet Détails.

   Dans la section Balises, toutes les balises du magasin de données sont répertoriées.

AWS CLI et SDK

CLI

AWS CLI

Exemple 1 : pour répertorier les balises de ressources d'un magasin de données

L'exemple de `list-tags-for-resource` code suivant répertorie les balises d'un magasin de données.

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

Sortie :

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

Exemple 2 : pour répertorier les balises de ressources pour un ensemble d'images

L'exemple de `list-tags-for-resource` code suivant répertorie les balises d'un ensemble d'images.

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

Sortie :

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

Pour plus d'informations, consultez la section Marquage des ressources avec AWS HealthImaging dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous ListTagsForResourceà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous ListTagsForResourceà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListTagsForResourceà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListTagsForResourcele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Débalisage d'une ressource

Utilisez cette `UntagResource`action pour annuler le balisage d'une ressource dans AWS HealthImaging. Les exemples de code suivants décrivent comment utiliser l'`UntagResource`action avec les AWS SDK AWS Management Console AWS CLI,, et. Pour plus d'informations, consultez la section Marquage de vos AWS ressources dans le Références générales AWS guide.

Pour annuler le balisage d'une ressource (magasin de données)

Choisissez un menu en fonction de vos préférences d'accès à AWS HealthImaging.

AWS Console

1. Ouvrez la page HealthImaging Stockages de données de la console.
2. Choisissez un magasin de données.

   La page de détails du magasin de données s'ouvre.
3. Cliquez sur l'onglet Détails.
4. Dans la section Balises, choisissez Gérer les balises.

   La page Gérer les balises s'ouvre.
5. Choisissez Supprimer à côté du tag que vous souhaitez supprimer.
6. Sélectionnez Enregistrer les modifications.

AWS CLI et SDK

CLI

AWS CLI

Exemple 1 : pour supprimer le balisage d'un magasin de données

L'exemple de `untag-resource` code suivant supprime les balises d'un magasin de données.

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

Cette commande ne produit aucun résultat.

Exemple 2 : pour annuler le balisage d'un ensemble d'images

L'exemple de `untag-resource` code suivant supprime les balises d'un ensemble d'images.

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

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Marquage des ressources avec AWS HealthImaging dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous UntagResourceà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
```

```
    }
```

- Pour plus de détails sur l'API, reportez-vous UntagResourceà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous UntagResourceà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez UntagResourcele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exemples de code pour HealthImaging l'utilisation des AWS SDK

Les exemples de code suivants montrent comment utiliser HealthImaging un kit de développement AWS logiciel (SDK).

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Mise en route

Bonjour HealthImaging

Les exemples de code suivants montrent comment commencer à utiliser HealthImaging.

C++

SDK pour C++

Code pour le MakeLists fichier CMake C.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})
```

Code du fichier source 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;
}
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"))
```

- Pour plus de détails sur l'API, consultez ListDatastoresle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exemples de code

Actions relatives à HealthImaging l'utilisation des AWS SDK

Les exemples de code suivants montrent comment effectuer des HealthImaging actions individuelles avec AWS les SDK. Ces extraits appellent l' HealthImaging API et sont des extraits de code de programmes plus volumineux qui doivent être exécutés en contexte. Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions pour configurer et exécuter le code.

Les exemples suivants incluent uniquement les actions les plus couramment utilisées. Pour obtenir la liste complète, veuillez consulter la AWS HealthImaging Référence d'API.

Exemples

- Utilisation CopyImageSet avec un AWS SDK ou une CLI
- Utilisation CreateDatastore avec un AWS SDK ou une CLI
- Utilisation DeleteDatastore avec un AWS SDK ou une CLI
- Utilisation DeleteImageSet avec un AWS SDK ou une CLI
- Utilisation GetDICOMImportJob avec un AWS SDK ou une CLI
- Utilisation GetDatastore avec un AWS SDK ou une CLI
- Utilisation GetImageFrame avec un AWS SDK ou une CLI
- Utilisation GetImageSet avec un AWS SDK ou une CLI
- Utilisation GetImageSetMetadata avec un AWS SDK ou une CLI
- Utilisation ListDICOMImportJobs avec un AWS SDK ou une CLI
- Utilisation ListDatastores avec un AWS SDK ou une CLI
- Utilisation ListImageSetVersions avec un AWS SDK ou une CLI
- Utilisation ListTagsForResource avec un AWS SDK ou une CLI
- Utilisation SearchImageSets avec un AWS SDK ou une CLI
- Utilisation StartDICOMImportJob avec un AWS SDK ou une CLI
- Utilisation TagResource avec un AWS SDK ou une CLI
- Utilisation UntagResource avec un AWS SDK ou une CLI
- Utilisation UpdateImageSetMetadata avec un AWS SDK ou une CLI

Utilisation **`CopyImageSet`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`CopyImageSet`.

CLI

AWS CLI

Exemple 1 : pour copier une série d'images sans destination.

L'exemple de `copy-image-set` code suivant crée une copie dupliquée d'un ensemble d'images sans destination.

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

Sortie :

```
{
    "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"
}
```

Exemple 2 : pour copier une série d'images avec une destination.

L'exemple de `copy-image-set` code suivant crée une copie dupliquée d'un ensemble d'images avec une destination.

```
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"} }'
```

Sortie :

```
{
```

```
    "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"
}
```

Pour plus d'informations, consultez la section Copier un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS CLI commandes.

Java

SDK pour 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 "";
    }
```

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Fonction utilitaire pour copier un ensemble d'images.

```
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;
};
```

Copiez un ensemble d'images sans destination.

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

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

Copiez un ensemble d'images avec une destination.

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

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Fonction utilitaire pour copier un ensemble d'images.

```
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"]
```

Copiez un ensemble d'images sans destination.

```
            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,
            )
```

Copiez un ensemble d'images avec une destination.

```
            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,
            )
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez CopyImageSetle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`CreateDatastore`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`CreateDatastore`.

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, reportez-vous CreateDatastoreà la section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour créer un magasin de données

L'exemple de `create-datastore` code suivant crée un magasin de données portant le nom`my-datastore`.

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

Sortie :

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

Pour plus d'informations, consultez la section Création d'un magasin de données dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous CreateDatastoreà la section Référence des AWS CLI commandes.

Java

SDK pour 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 "";
   }
```

- Pour plus de détails sur l'API, reportez-vous CreateDatastoreà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous CreateDatastoreà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez CreateDatastorele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`DeleteDatastore`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`DeleteDatastore`.

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteDatastoreà la section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour supprimer un magasin de données

L'exemple de `delete-datastore` code suivant supprime un magasin de données.

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

Sortie :

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

Pour plus d'informations, consultez la section Suppression d'un magasin de données dans le guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous DeleteDatastoreà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteDatastoreà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteDatastoreà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez DeleteDatastorele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`DeleteImageSet`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`DeleteImageSet`.

Les exemples d’actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l’exemple de code suivant :

- Commencez avec les ensembles d'images et les cadres d'images

C++

SDK pour 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();
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour supprimer un ensemble d'images

L'exemple de `delete-image-set` code suivant supprime un ensemble d'images.

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

Sortie :

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

Pour plus d'informations, consultez la section Suppression d'un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez DeleteImageSetle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`GetDICOMImportJob`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`GetDICOMImportJob`.

Les exemples d’actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l’exemple de code suivant :

- Commencez avec les ensembles d'images et les cadres d'images

C++

SDK pour 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;
}
```

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans la section AWS SDK for C++ API Reference.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour obtenir les propriétés d'une tâche d'importation DICOM

L'exemple de `get-dicom-import-job` code suivant permet d'obtenir les propriétés d'une tâche d'importation DICOM.

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

Sortie :

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

Pour plus d'informations, consultez la section Obtenir les propriétés des tâches d'importation dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans AWS CLI Command Reference.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans la section AWS SDK for Java 2.x API Reference.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans la section AWS SDK for JavaScript API Reference.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

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

- Pour plus de détails sur l'API, consultez GetDICOM ImportJob dans le manuel de référence de l'API AWS SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`GetDatastore`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`GetDatastore`.

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour obtenir les propriétés d'un magasin de données

L'exemple de `get-datastore` code suivant permet d'obtenir les propriétés d'un magasin de données.

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

Sortie :

```
{
    "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"
    }
}
```

Pour plus d'informations, consultez la section Obtenir les propriétés du magasin de données dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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"];
};
```

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetDatastorele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`GetImageFrame`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`GetImageFrame`.

Les exemples d’actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l’exemple de code suivant :

- Commencez avec les ensembles d'images et les cadres d'images

C++

SDK pour 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();
}
```

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour obtenir les données relatives aux pixels définis par image

L'exemple de `get-image-frame` code suivant permet d'obtenir un cadre d'image.

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

Remarque : Cet exemple de code n'inclut pas de sortie car l' GetImageFrame action renvoie un flux de données de pixels vers le fichier imageframe.jph. Pour plus d'informations sur le décodage et l'affichage de trames d'image, consultez la section Bibliothèques de décodage HTJ2K.

Pour plus d'informations, consultez la section Obtenir des données en pixels d'un ensemble d'images dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
               }
       }
```

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetImageFramele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`GetImageSet`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`GetImageSet`.

CLI

AWS CLI

Pour obtenir les propriétés des ensembles d'images

L'exemple de `get-image-set` code suivant permet d'obtenir les propriétés d'un ensemble d'images.

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

Sortie :

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

Pour plus d'informations, consultez la section Obtenir les propriétés d'un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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 = "xxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxx",
  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: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetImageSetle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`GetImageSetMetadata`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`GetImageSetMetadata`.

Les exemples d’actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l’exemple de code suivant :

- Commencez avec les ensembles d'images et les cadres d'images

C++

SDK pour C++

Fonction utilitaire pour obtenir les métadonnées d'un ensemble d'images.

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

Obtenez les métadonnées des ensembles d'images sans version.

```
        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;
        }
```

Obtenez les métadonnées des ensembles d'images avec la version.

```
        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;
        }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Exemple 1 : Pour obtenir les métadonnées d'un ensemble d'images sans version

L'exemple de `get-image-set-metadata` code suivant permet d'obtenir les métadonnées d'un ensemble d'images sans spécifier de version.

Remarque : `outfile` est un paramètre obligatoire

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

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

Les métadonnées renvoyées sont compressées avec gzip et stockées dans le fichier studymetadata.json.gz. Pour visualiser le contenu de l'objet JSON renvoyé, vous devez d'abord le décompresser.

Sortie :

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

Exemple 2 : Pour obtenir les métadonnées d'un ensemble d'images avec la version

L'exemple de `get-image-set-metadata` code suivant permet d'obtenir les métadonnées d'un ensemble d'images avec une version spécifiée.

Remarque : `outfile` est un paramètre obligatoire

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

Les métadonnées renvoyées sont compressées avec gzip et stockées dans le fichier studymetadata.json.gz. Pour visualiser le contenu de l'objet JSON renvoyé, vous devez d'abord le décompresser.

Sortie :

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

Pour plus d'informations, consultez la section Obtenir les métadonnées d'un ensemble d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
   }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Fonction utilitaire pour obtenir les métadonnées d'un ensemble d'images.

```
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;
};
```

Obtenez les métadonnées des ensembles d'images sans version.

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

Obtenez les métadonnées des ensembles d'images avec la version.

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

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Fonction utilitaire pour obtenir les métadonnées d'un ensemble d'images.

```
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
```

Obtenez les métadonnées des ensembles d'images sans version.

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

Obtenez les métadonnées des ensembles d'images avec la version.

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

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez GetImageSetMetadatale AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`ListDICOMImportJobs`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`ListDICOMImportJobs`.

CLI

AWS CLI

Pour répertorier les tâches d'importation Dicom

L'exemple de `list-dicom-import-jobs` code suivant répertorie les tâches d'importation dicom.

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

Sortie :

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

Pour plus d'informations, consultez la section Répertorier les tâches d'importation dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, voir ListDicom ImportJobs dans AWS CLI Command Reference.

Java

SDK pour 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<>();
    }
```

- Pour plus de détails sur l'API, voir ListDicom ImportJobs dans la référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, voir ListDicom ImportJobs dans la référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListDICOM ImportJobs dans le manuel de référence de l'API AWS SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **ListDatastores** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliserListDatastores.

Bash

AWS CLI avec le 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
}
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS CLI commandes.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour répertorier les magasins de données

L'exemple de `list-datastores` code suivant répertorie les magasins de données disponibles.

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

Sortie :

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

Pour plus d'informations, consultez la section Répertorier les banques de données dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListDatastoresle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`ListImageSetVersions`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`ListImageSetVersions`.

CLI

AWS CLI

Pour répertorier les versions d'ensembles d'images

L'exemple de `list-image-set-versions` code suivant répertorie l'historique des versions d'un ensemble d'images.

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

Sortie :

```
{
    "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
        }
    ]
}
```


Pour plus d'informations, consultez la section Répertorier les versions des ensembles d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListImageSetVersionsle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`ListTagsForResource`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`ListTagsForResource`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Marquage d'un magasin de données
- Marquer un ensemble d'images

CLI

AWS CLI

Exemple 1 : pour répertorier les balises de ressources d'un magasin de données

L'exemple de `list-tags-for-resource` code suivant répertorie les balises d'un magasin de données.

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

Sortie :

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

Exemple 2 : pour répertorier les balises de ressources pour un ensemble d'images

L'exemple de `list-tags-for-resource` code suivant répertorie les balises d'un ensemble d'images.

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

Sortie :

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

Pour plus d'informations, consultez la section Marquage des ressources avec AWS HealthImaging dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous ListTagsForResourceà la section Référence des AWS CLI commandes.

Java

SDK pour 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous ListTagsForResourceà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListTagsForResourceà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez ListTagsForResourcele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`SearchImageSets`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`SearchImageSets`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Commencez avec les ensembles d'images et les cadres d'images

C++

SDK pour C++

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
//! 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;
}
```

Cas d'utilisation #1 : opérateur 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;
           }
       }
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
        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;
           }
       }
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
       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;
           }
       }
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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;
            }
        }
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Exemple 1 : Pour rechercher des ensembles d'images à l'aide d'un opérateur EQUAL

L'exemple de `search-image-sets` code suivant utilise l'opérateur EQUAL pour rechercher des ensembles d'images en fonction d'une valeur spécifique.

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

Contenu de `search-criteria.json`

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

Sortie :

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

Exemple 2 : Pour rechercher des ensembles d'images avec un opérateur BETWEEN à l'aide de DICOM StudyDate et DICOM StudyTime

L'exemple de `search-image-sets` code suivant recherche des ensembles d'images contenant des études DICOM générées entre le 1er janvier 1990 (00h00) et le 1er janvier 2023 (00h00).

Remarque : le format DICOM StudyTime est facultatif. S'il n'est pas présent, 00 h 00 (début de journée) est la valeur horaire pour les dates fournies pour le filtrage.

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

Contenu de `search-criteria.json`

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

Sortie :

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

Exemple 3 : Pour rechercher des ensembles d'images avec un opérateur BETWEEN à l'aide de CreatedAt (les études temporelles étaient précédemment conservées)

L'exemple de `search-image-sets` code suivant recherche des ensembles d'images contenant des études DICOM persistantes HealthImaging entre les plages horaires du fuseau horaire UTC.

Remarque : Indiquez CreatedAt dans le format d'exemple (« 1985-04-12T 23:20:50,52 Z »).

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

Contenu de `search-criteria.json`

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

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

Sortie :

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

Exemple 4 : Pour rechercher des ensembles d'images avec un opérateur EQUAL sur DICOM SeriesInstance UID et BETWEEN sur UpdateDat et trier les réponses dans l'ordre ASC sur le champ UpdatedAt

L'exemple de `search-image-sets` code suivant recherche des ensembles d'images avec un opérateur EQUAL sur DICOM SeriesInstance UID et BETWEEN sur UpdateDat et trie la réponse dans l'ordre ASC sur le champ UpdatedAt.

Remarque : Fournissez UpdateDat dans le format d'exemple (« 1985-04-12T 23:20:50,52 Z »).

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

Contenu de `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"
    }
}
```

Sortie :

```
{
    "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"
```

```
    }]
}
```

Pour plus d'informations, consultez la section Recherche de séries d'images dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
    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;
    }
```

Cas d'utilisation #1 : opérateur 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();
        }
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
        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();
        }
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
        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();
        }
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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();
        }
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
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;
};
```

Cas d'utilisation #1 : opérateur EQUAL.

```
    const datastoreId = "12345678901234567890123456789012";

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

```
            },
        ]
    };

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

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
const datastoreId = "1234567890123456789012345678901 2";

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);
}
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
    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);
    }
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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);
    }
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Fonction utilitaire permettant de rechercher des ensembles d'images.

```
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
```

Cas d'utilisation #1 : opérateur 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}")
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
        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}"
        )
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
        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}"
        )
```

Cas d'utilisation #4 : opérateur EQUAL sur l'SeriesInstanceUID DICOM et BETWEEN sur UpdateDat et tri la réponse dans l'ordre ASC sur le champ 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}")
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez SearchImageSetsle AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`StartDICOMImportJob`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`StartDICOMImportJob`.

Les exemples d’actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l’exemple de code suivant :

- Commencez avec les ensembles d'images et les cadres d'images

C++

SDK pour 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();
}
```

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans la référence des AWS SDK for C++ API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

CLI

AWS CLI

Pour démarrer une tâche d'importation DICOM

L'exemple de `start-dicom-import-job` code suivant démarre une tâche d'importation 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"
```

Sortie :

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

Pour plus d'informations, consultez la section Démarrage d'une tâche d'importation dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans AWS CLI Command Reference.

Java

SDK pour 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 "";
   }
```

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans la référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, voir StartDICOM ImportJob dans la référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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"]
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez StartDICOM ImportJob dans le manuel de référence de l'API AWS SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`TagResource`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`TagResource`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Marquage d'un magasin de données
- Marquer un ensemble d'images

CLI

AWS CLI

Exemple 1 : pour étiqueter un magasin de données

Les exemples de `tag-resource` code suivants balisent un magasin de données.

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

Cette commande ne produit aucun résultat.

Exemple 2 : pour baliser un ensemble d'images

Les exemples de `tag-resource` code suivants balisent un ensemble d'images.

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

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Marquage des ressources avec AWS HealthImaging dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
   }
```

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez TagResourcele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`UntagResource`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`UntagResource`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Marquage d'un magasin de données
- Marquer un ensemble d'images

CLI

AWS CLI

Exemple 1 : pour supprimer le balisage d'un magasin de données

L'exemple de `untag-resource` code suivant supprime les balises d'un magasin de données.

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

Cette commande ne produit aucun résultat.

Exemple 2 : pour annuler le balisage d'un ensemble d'images

L'exemple de `untag-resource` code suivant supprime les balises d'un ensemble d'images.

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

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Marquage des ressources avec AWS HealthImaging dans le Guide du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous UntagResourceà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
       }
   }
```

- Pour plus de détails sur l'API, reportez-vous UntagResourceà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
```

```
};
```

- Pour plus de détails sur l'API, reportez-vous UntagResourceà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus de détails sur l'API, consultez UntagResourcele AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`UpdateImageSetMetadata`** avec un AWS SDK ou une CLI

Les exemples de code suivants montrent comment utiliser`UpdateImageSetMetadata`.

CLI

AWS CLI

Pour insérer ou mettre à jour un attribut dans les métadonnées d'un ensemble d'images

L'exemple de `update-image-set-metadata` code suivant insère ou met à jour un attribut dans les métadonnées du jeu d'images.

```
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
```

Contenu de `metadata-updates.json`

```
{
```

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

Remarque : `updatableAttributes` est une chaîne JSON codée en Base64. Voici la chaîne JSON non codée.

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

Sortie :

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

Pour supprimer un attribut des métadonnées d'un ensemble d'images

L'exemple de `update-image-set-metadata` code suivant supprime un attribut des métadonnées du jeu d'images.

```
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
```

Contenu de `metadata-updates.json`

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

```
}
```

Remarque : `removableAttributes` est une chaîne JSON codée en Base64. Voici la chaîne JSON non codée. La clé et la valeur doivent correspondre à l'attribut à supprimer.

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

Sortie :

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

Pour supprimer une instance des métadonnées du jeu d'images

L'exemple de `update-image-set-metadata` code suivant supprime une instance des métadonnées du jeu d'images.

```
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
```

Contenu de `metadata-updates.json`

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

Remarque : `removableAttributes` est une chaîne JSON codée en Base64. Voici la chaîne JSON non codée.

{« 1.1.1.1.1.1.12345.123456789012.123.12345678901234.1" : {" Instances » : {"1.1.1.1.1.12345.123456789012.123.12345678901234.1" : {}}}}

Sortie :

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

Pour plus d'informations, consultez la section Mise à jour des métadonnées des ensembles d'images dans le manuel du AWS HealthImaging développeur.

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS CLI commandes.

Java

SDK pour 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);
        }
    }
```

Cas d'utilisation #1 : Insérer ou mettre à jour un attribut.

```
            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);
```

Cas d'utilisation #2 : Supprimer un attribut.

```
            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);
```

Cas d'utilisation #3 : Supprimer une instance.

```
          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);
```

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS SDK for Java 2.x API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour 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;
};
```

Cas d'utilisation #1 : Insérer ou mettre à jour un attribut.

```
        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);
```

Cas d'utilisation #2 : Supprimer un attribut.

```
        // 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);
```

Cas d'utilisation #3 : Supprimer une instance.

```
        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);
```

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS SDK for JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour 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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

Cas d'utilisation #1 : Insérer ou mettre à jour un attribut.

```
            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
            )
```

Cas d'utilisation #2 : Supprimer un attribut.

```
            # 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
            )
```

Cas d'utilisation #3 : Supprimer une instance.

```
            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
            )
```

- Pour plus de détails sur l'API, consultez UpdateImageSetMetadatale AWS manuel de référence de l'API SDK for Python (Boto3).

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Scénarios d' HealthImaging utilisation des AWS SDK

Les exemples de code suivants vous montrent comment implémenter des scénarios courants HealthImaging avec AWS les SDK. Ces scénarios vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions HealthImaging. Chaque scénario inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code.

Exemples

- Commencez à utiliser des ensembles HealthImaging d'images et des cadres d'images à l'aide d'un AWS SDK
- Marquage d'un magasin de HealthImaging données à l'aide d'un SDK AWS
- Marquage d'un ensemble d' HealthImaging images à l'aide d'un SDK AWS

Commencez à utiliser des ensembles HealthImaging d'images et des cadres d'images à l'aide d'un AWS SDK

Les exemples de code suivants montrent comment importer des fichiers DICOM et télécharger des cadres d'image dans HealthImaging.

L'implémentation est structurée comme une application de ligne de commande de flux de travail.

- Configurez les ressources pour une importation DICOM.
- Importez des fichiers DICOM dans un magasin de données.

- Récupérez les identifiants des ensembles d'images pour la tâche d'importation.
- Récupérez les identifiants des cadres d'image pour les ensembles d'images.
- Téléchargez, décodez et vérifiez les cadres d'image.
- Nettoyez les ressources.

C++

Kit de développement logiciel (SDK) for C++

Créez une AWS CloudFormation pile avec les ressources nécessaires.

```
   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: ");
    }
```

Copiez les fichiers DICOM dans le compartiment d'importation 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;
    }
```

Importez les fichiers DICOM dans le magasin de données 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;
}
```

Obtenez les ensembles d'images créés par la tâche d'importation 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;
}
```

Obtenez des informations sur les cadres d'image pour les ensembles d'images.

```
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();
}
```

Téléchargez, décodez et vérifiez les cadres d'image.

```
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;
}
```

Nettoyez les ressources.

```
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;
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for C++ .
  - DeleteImageSet
  - Obtenez DICOM ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Démarrez DICOM ImportJob

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

index.js- Orchestrez les étapes.

```
// 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- Déployez des ressources.

```
// 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- Copiez des fichiers 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- Lancez l'importation dans la banque de données.

```
// 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- Obtenez les identifiants des ensembles d'images.

```
// 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- Obtenez les identifiants des cadres d'image.

```
// 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- Vérifiez les cadres d'image. La bibliothèque AWS HealthImaging Pixel Data Verification a été utilisée pour la vérification.

```
// 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- Détruisez des ressources.

```
// 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,
  },
);
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - DeleteImageSet
  - Obtenez DICOM ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Démarrez DICOM ImportJob

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Créez une AWS CloudFormation pile avec les ressources nécessaires.

```
    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
```

Copiez les fichiers DICOM dans le compartiment d'importation 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.")
```

Importez les fichiers DICOM dans le magasin de données 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"]
```

Obtenez les ensembles d'images créés par la tâche d'importation 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
```

Obtenez des informations sur les cadres d'image pour les ensembles d'images.

```
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
```

Téléchargez, décodez et vérifiez les cadres d'image.

```
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
```

Nettoyez les ressources.

```
    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
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans AWS SDK for Python (Boto3) API Reference.
  - DeleteImageSet
  - Obtenez DICOM ImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Démarrez DICOM ImportJob

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Marquage d'un magasin de HealthImaging données à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment étiqueter un magasin de HealthImaging données.

Java

SDK pour Java 2.x

Pour étiqueter un magasin de données.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                ImmutableMap.of("Deployment", "Development"));
```

Fonction utilitaire permettant de baliser une ressource.

```
   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);
       }
   }
```

Pour répertorier les balises d'un magasin de données.

```
                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());
                }
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
   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;
   }
```

Pour supprimer le balisage d'un magasin de données.

```
                final String datastoreArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 datastoreArn,
                                Collections.singletonList("Deployment"));
```

Fonction utilitaire permettant de débaliser une ressource.

```
   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);
       }
   }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Pour étiqueter un magasin de données.

```
 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);
  }
```

Fonction utilitaire permettant de baliser une ressource.

```
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;
};
```

Pour répertorier les balises d'un magasin de données.

```
  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);
  }
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
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;
};
```

Pour supprimer le balisage d'un magasin de données.

```
  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);
  }
```

Fonction utilitaire permettant de débaliser une ressource.

```
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;
};
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Pour étiqueter un magasin de données.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.tag_resource(data_store_arn, {"Deployment":
 "Development"})
```

Fonction utilitaire permettant de baliser une ressource.

```
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
```

Pour répertorier les balises d'un magasin de données.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.list_tags_for_resource(data_store_arn)
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
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"]
```

Pour supprimer le balisage d'un magasin de données.

```
    a_data_store_arn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012"

    medical_imaging_wrapper.untag_resource(data_store_arn, ["Deployment"])
```

Fonction utilitaire permettant de débaliser une ressource.

```
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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans AWS SDK for Python (Boto3) API Reference.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Marquage d'un ensemble d' HealthImaging images à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment baliser un ensemble HealthImaging d'images.

Java

SDK pour Java 2.x

Pour baliser un ensemble d'images.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                TagResource.tagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                ImmutableMap.of("Deployment", "Development"));
```

Fonction utilitaire permettant de baliser une ressource.

```
    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);
        }
    }
```

Pour répertorier les balises d'un ensemble d'images.

```
                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());
                }
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
   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;
   }
```

Pour annuler le balisage d'un ensemble d'images.

```
                final String imageSetArn = "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";

                UntagResource.untagMedicalImagingResource(medicalImagingClient,
 imageSetArn,
                                Collections.singletonList("Deployment"));
```

Fonction utilitaire permettant de débaliser une ressource.

```
   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);
       }
   }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

JavaScript

SDK pour JavaScript (v3)

Pour baliser un ensemble d'images.

```
  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);
  }
```

Fonction utilitaire permettant de baliser une ressource.

```
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;
};
```

Pour répertorier les balises d'un ensemble d'images.

```
  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);
  }
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
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;
};
```

Pour annuler le balisage d'un ensemble d'images.

```
  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);
  }
```

Fonction utilitaire permettant de débaliser une ressource.

```
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;
};
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Python

SDK pour Python (Boto3)

Pour baliser un ensemble d'images.

```
    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"})
```

Fonction utilitaire permettant de baliser une ressource.

```
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
```

Pour répertorier les balises d'un ensemble d'images.

```
    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)
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
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"]
```

Pour annuler le balisage d'un ensemble d'images.

```
    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"])
```

Fonction utilitaire permettant de débaliser une ressource.

```
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
```

Le code suivant instancie l' MedicalImagingWrapper objet.

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

- Pour plus d’informations sur l’API, consultez les rubriques suivantes dans AWS SDK for Python (Boto3) API Reference.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation HealthImaging avec un AWS SDK. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Surveillance d'AWS HealthImaging

La surveillance et la journalisation jouent un rôle important dans le maintien de la sécurité, de la fiabilité, de la disponibilité et des performances d'AWS HealthImaging. AWS fournit les outils de journalisation et de surveillance suivants pour surveiller HealthImaging, signaler tout problème et prendre des mesures automatiques le cas échéant :

- AWS CloudTrailcapture les appels d'API et les événements associés effectués par ou pour le compte de votre AWS compte et envoie les fichiers journaux dans un compartiment Amazon S3 que vous spécifiez. Vous pouvez identifier les utilisateurs et les comptes appelés AWS, l'adresse IP source à partir de laquelle les appels ont été effectués et la date des appels. Pour plus d'informations, consultez le Guide de l'utilisateur AWS CloudTrail.
- Amazon CloudWatch surveille vos AWS ressources et les applications que vous utilisez AWS en temps réel. Vous pouvez collecter et suivre les métriques, créer des tableaux de bord personnalisés, et définir des alarmes qui vous informent ou prennent des mesures lorsqu'une métrique spécifique atteint un seuil que vous spécifiez. Par exemple, vous pouvez CloudWatch suivre l'utilisation du processeur ou d'autres indicateurs de vos instances Amazon EC2 et lancer automatiquement de nouvelles instances en cas de besoin. Pour plus d'informations, consultez le guide de CloudWatch l'utilisateur Amazon.
- Amazon EventBridge est un service de bus d'événements sans serveur qui permet de connecter facilement vos applications à des données provenant de diverses sources. EventBridge fournit un flux de données en temps réel à partir de vos propres applications, applications oftware-as-a S-Service (SaaS) et AWS services et achemine ces données vers des cibles telles que Lambda. Cela vous permet de surveiller les événements qui se produisent dans les services et de créer des architectures basées sur les événements. Pour plus d'informations, consultez le guide de EventBridge l'utilisateur Amazon.

Rubriques

Utilisation AWS CloudTrail avec HealthImaging

AWS HealthImaging est intégré à AWS CloudTrail un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un AWS service dans HealthImaging. CloudTrail capture tous les appels d'API HealthImaging sous forme d'événements. Les appels capturés incluent des appels provenant de la HealthImaging console et des appels de code vers les opérations de l' HealthImaging API. Si vous créez un suivi, vous pouvez activer la diffusion continue des CloudTrail événements vers un compartiment Amazon S3, y compris les événements pour HealthImaging. Si vous ne configurez pas de suivi, vous pouvez toujours consulter les événements les plus récents dans la CloudTrail console dans Historique des événements. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été faite HealthImaging, l'adresse IP à partir de laquelle la demande a été faite, qui a fait la demande, quand elle a été faite et des détails supplémentaires.

Pour en savoir plus CloudTrail, consultez le guide de AWS CloudTrail l'utilisateur.

Création d’un journal de suivi

CloudTrail est activé pour vous Compte AWS lorsque vous créez le compte. Lorsqu'une activité se produit dans HealthImaging, cette activité est enregistrée dans un CloudTrail événement avec d'autres événements de AWS service dans l'historique des événements. Vous pouvez consulter, rechercher et télécharger les événements récents dans votre Compte AWS. Pour plus d'informations, consultez la section Affichage des événements avec l'historique des CloudTrail événements.

> **Note**
>
> Pour afficher l'historique des CloudTrail événements pour AWS HealthImaging dans le AWS Management Console, accédez au menu des attributs de recherche, sélectionnez Source de l'événement, puis choisissez`medical-imaging.amazonaws.com`.

Pour un enregistrement continu des événements de votre région Compte AWS, y compris des événements pour HealthImaging, créez un parcours. Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Par défaut, lorsque vous créez un journal d’activité dans la console, il s’applique à toutes les régions Régions AWS. Le journal enregistre les événements de toutes les régions de la AWS partition et transmet les fichiers journaux au compartiment Amazon S3 que vous spécifiez. En outre, vous pouvez configurer d'autres AWS

services pour analyser plus en détail les données d'événements collectées dans les CloudTrail journaux et agir en conséquence. Pour plus d'informations, consultez les ressources suivantes :

- Présentation de la création d'un journal de suivi
- CloudTrail services et intégrations pris en charge
- Configuration des notifications Amazon SNS pour CloudTrail
- Réception de fichiers CloudTrail journaux de plusieurs régions et réception de fichiers CloudTrail journaux de plusieurs comptes

> **Note**
>
> AWS HealthImaging prend en charge deux types d' CloudTrail événements : les événements de gestion et les événements liés aux données. Les événements de gestion sont les événements généraux générés par chaque AWS service, notamment HealthImaging.
> Par défaut, la journalisation est appliquée aux événements de gestion pour chaque appel HealthImaging d'API pour lequel elle est activée. Les événements liés aux données sont facturables et généralement réservés aux API dont le nombre de transactions par seconde (tps) est élevé. Vous pouvez donc choisir de ne pas avoir de CloudTrail journaux pour des raisons de coût.
> Avec HealthImaging, toutes les actions d'API répertoriées dans la référence des HealthImaging API AWS sont considérées comme des événements de gestion, à l'exception de `GetImageFrame`. L'`GetImageFrame`action est intégrée en CloudTrail tant qu'événement de données et doit donc être activée. Pour plus d'informations, consultez Journalisation des événements de données dans le Guide de l'utilisateur AWS CloudTrail .

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer les éléments suivants :

- Si la demande a été faite avec les informations d'identification de l'utilisateur root ou AWS Identity and Access Management (IAM).
- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la demande a été faite par un autre AWS service.

Pour plus d'informations, consultez l'CloudTrail `userIdentity`élément.

Comprendre les entrées du journal

Un suivi est une configuration qui permet de transmettre des événements sous forme de fichiers journaux à un compartiment Amazon S3 que vous spécifiez. CloudTrail les fichiers journaux contiennent une ou plusieurs entrées de journal. Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'action demandée, la date et l'heure de l'action, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics, ils n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant montre une entrée de CloudTrail journal HealthImaging illustrant l'`GetDICOMImportJob`action.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "XXXXXXXXXXXXXXXXXXXXX:ce6d90ba-5fba-4456-a7bc-f9bc877597c3",
        "arn": "arn:aws:sts::123456789012:assumed-role/TestAccessRole/
ce6d90ba-5fba-4456-a7bc-f9bc877597c3"
        "accountId": "123456789012",
        "accessKeyId": "XXXXXXXXXXXXXXXXXXX",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "XXXXXXXXXXXXXXXXXXXXX",
                "arn": "arn:aws:iam::123456789012:role/TestAccessRole",
                "accountId": "123456789012",
                "userName": "TestAccessRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-28T15:52:42Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-28T16:02:30Z",
    "eventSource": "medical-imaging.amazonaws.com",
    "eventName": "GetDICOMImportJob",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
```

```
    "userAgent": "aws-sdk-java/2.18.1 Linux/5.4.209-129.367.amzn2int.x86_64 OpenJDK_64-
Bit_Server_VM/11.0.17+9-LTS Java/11.0.17 vendor/Amazon.com_Inc. md/internal io/sync
 http/Apache cfg/retry-mode/standard",
    "requestParameters": {
        "jobId": "5d08d05d6aab2a27922d6260926077d4",
        "datastoreId": "12345678901234567890123456789012"
    },
    "responseElements": null,
    "requestID": "922f5304-b39f-4034-9d2e-f062de092a44",
    "eventID": "26307f73-07f4-4276-b379-d362aa303b22",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "824333766656",
    "eventCategory": "Management"
}
```


Utiliser Amazon CloudWatch avec HealthImaging

Vous pouvez surveiller AWS HealthImaging à l'aide d'AWS CloudWatch, qui collecte les données brutes et les traite en métriques lisibles en temps quasi réel. Ces statistiques sont conservées pendant 15 mois, afin que vous puissiez utiliser ces informations historiques et avoir une meilleure idée des performances de votre application ou service Web. Vous pouvez également définir des alarmes qui surveillent certains seuils et envoient des notifications ou prennent des mesures lorsque ces seuils sont atteints. Pour plus d'informations, consultez le guide de CloudWatch l'utilisateur Amazon.

> **Note**
>
> Les métriques sont signalées pour toutes les HealthImaging API.

Les tableaux suivants répertorient les mesures et les dimensions pour HealthImaging. Chacun est présenté sous forme de comptage de fréquences pour une plage de données spécifiée par l'utilisateur.

Métriques

Métriques	Description
Nombre d'appels	Le nombre d'appels aux API. Cela peut être signalé soit pour le compte, soit pour un magasin de données spécifique. Unités : nombre Statistiques valides : somme, nombre Dimensions : fonctionnement, identifiant du magasin de données, type de magasin de données

Vous pouvez obtenir des métriques pour HealthImaging AWS Management Console AWS CLI, le ou l' CloudWatch API. Vous pouvez utiliser l' CloudWatch API via l'un des kits de développement logiciel (SDK) Amazon AWS ou les outils CloudWatch d'API. La HealthImaging console affiche des graphiques basés sur les données brutes de l' CloudWatch API.

Vous devez disposer des CloudWatch autorisations appropriées pour effectuer la surveillance HealthImaging CloudWatch. Pour plus d'informations, consultez la section Gestion des identités et des accès CloudWatch dans le guide de CloudWatch l'utilisateur.

Afficher HealthImaging les métriques

Pour consulter les métriques (CloudWatch console)

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la.
2. Choisissez Metrics, choisissez All Metrics, puis choisissez AWS/Medical Imaging.
3. Choisissez la dimension, le nom de la métrique, puis Ajouter au graphique.
4. Choisissez une valeur pour la plage de dates. Le décompte de la métrique pour la plage de dates sélectionnée est affiché dans le graphique.

Création d'une alarme à l'aide de CloudWatch

Une CloudWatch alarme surveille une seule métrique sur une période spécifiée et exécute une ou plusieurs actions : envoyer une notification à une rubrique Amazon Simple Notification Service (Amazon SNS) ou à une politique Auto Scaling. L'action ou les actions sont basées sur la valeur de la métrique par rapport à un seuil donné sur un certain nombre de périodes que vous spécifiez. CloudWatch peut également vous envoyer un message Amazon SNS lorsque l'alarme change d'état.

CloudWatch les alarmes appellent des actions uniquement lorsque l'état change et persiste pendant la période que vous spécifiez. Pour plus d'informations, consultez la section Utilisation des CloudWatch alarmes.

Utiliser Amazon EventBridge avec HealthImaging

Amazon EventBridge est un service sans serveur qui utilise des événements pour connecter les composants de l'application entre eux, ce qui vous permet de créer plus facilement des applications évolutives pilotées par des événements. La base EventBridge est de créer des règles qui acheminent les événements vers des cibles. AWS HealthImaging fournit une livraison durable des modifications d'état à EventBridge. Pour plus d'informations, consultez Qu'est-ce qu'Amazon EventBridge ? dans le guide de EventBridge l'utilisateur Amazon.

Rubriques

- HealthImaging événements envoyés à EventBridge
- HealthImaging structure d'événements et exemples

HealthImaging événements envoyés à EventBridge

Le tableau suivant répertorie tous les HealthImaging événements envoyés EventBridge pour traitement.

HealthImaging type d'événement	État
Événements liés à la banque de données	
Création d'un magasin de données	`CREATING`

HealthImaging type d'événement	État
Échec de la création du magasin de données	`CREATE_FAILED`
Stockage de données créé	`ACTIVE`
Suppression du magasin de données	`DELETING`
Stockage de données supprimé	`DELETED`
Pour plus d'informations, consultez DataStoreStatus dans le manuel de référence des API AWS HealthImaging .	
Importer des événements professionnels	
Importer un job soumis	`SUBMITTED`
Importer le Job en cours	`IN_PROGRESS`
Job d'importation terminé	`COMPLETED`
Échec de la tâche d'importation	`FAILED`
Pour plus d'informations, consultez JobStatus dans le manuel de référence des HealthImaging API AWS.	
Événements liés à une série d'images	
Ensemble d'images créé	`CREATED`
Copie d'un ensemble d'images	`COPYING`
Copie d'un ensemble d'images avec accès en lecture seule	`COPYING_WITH_READ_ONLY_ACCESS`
Ensemble d'images copié	`COPIED`
Échec de la copie du jeu d'images	`COPY_FAILED`
Mise à jour du jeu d'images	`UPDATING`

HealthImaging type d'événement	État
Ensemble d'images mis à jour	`UPDATED`
Échec de la mise à jour du jeu	`UPDATE_FAILED`
Suppression d'un ensemble d'images	`DELETING`
Ensemble d'images supprimé	`DELETED`

Pour plus d'informations, consultez ImageSetWorkflowStatusle manuel de référence des HealthImaging API AWS.

HealthImaging structure d'événements et exemples

HealthImaging les événements sont des objets dotés d'une structure JSON qui contiennent également des détails sur les métadonnées. Vous pouvez utiliser les métadonnées comme entrée pour recréer un événement ou obtenir plus d'informations. Tous les champs de métadonnées associés sont répertoriés dans un tableau sous les exemples de code dans les menus suivants. Pour plus d'informations, consultez la section Référence relative à la structure des événements dans le guide de EventBridge l'utilisateur Amazon.

> **Note**
>
> L'`source`attribut des structures HealthImaging d'événements est`aws.medical-imaging`.

Événements liés à la banque de données

Data Store Creating

État - **`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

État - **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

État - **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

État - **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

État - **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"
    }
}
```

Événements du magasin de données : descriptions des métadonnées

Nom	Type	Description
`version`	chaîne	Version du schéma d' EventBridge événements.
`id`	chaîne	L'UUID version 4 généré pour chaque événement.
`detail-type`	chaîne	Type d'événement envoyé.
`source`	chaîne	Identifie le service qui a généré l'événement.
`account`	chaîne	L'ID de compte AWS à 12 chiffres du propriétaire du magasin de données.
`time`	chaîne	Heure à laquelle l'événement s'est produit.
`region`	chaîne	Identifie la AWS région du magasin de données.

Nom	Type	Description
`resources`	tableau (chaîne)	Tableau JSON qui contient l'ARN du magasin de données.
`detail`	objet	Un objet JSON qui contient des informations sur l'événement.
`detail.imagingVersion`	chaîne	L'ID de version qui suit les modifications apportées au HealthImaging schéma détaillé des événements.
`detail.datastoreId`	chaîne	ID de banque de données associé à l'événement de changement de statut.
`detail.datastoreName`	chaîne	Le nom du magasin de données.
`detail.datastoreStatus`	chaîne	État actuel du magasin de données.

Importer des événements professionnels

Import Job Submitted

État - **`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

État - **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

État - **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

État - **`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/"
    }
```

```
}
```

Importer des événements professionnels : descriptions des métadonnées

Nom	Type	Description
`version`	chaîne	Version du schéma d' EventBridge événements.
`id`	chaîne	L'UUID version 4 généré pour chaque événement.
`detail-type`	chaîne	Type d'événement envoyé.
`source`	chaîne	Identifie le service qui a généré l'événement.
`account`	chaîne	L'ID de compte AWS à 12 chiffres du propriétaire du magasin de données.
`time`	chaîne	Heure à laquelle l'événement s'est produit.
`region`	chaîne	Identifie la AWS région du magasin de données.
`resources`	tableau (chaîne)	Tableau JSON qui contient l'ARN du magasin de données.
`detail`	objet	Un objet JSON qui contient des informations sur l'événement.
`detail.imagingVersion`	chaîne	L'ID de version qui suit les modifications apportées au HealthImaging schéma détaillé des événements.

Nom	Type	Description
`detail.datastoreId`	chaîne	Le magasin de données qui a généré l'événement de changement de statut.
`detail.jobId`	chaîne	ID de tâche d'importation associé à l'événement de changement de statut.
`detail.jobName`	chaîne	Le nom de la tâche d'importation.
`detail.jobStatus`	chaîne	Le statut actuel du poste.
`detail.inputS3Uri`	chaîne	Le chemin du préfixe d'entrée pour le compartiment S3 qui contient les fichiers DICOM à importer.
`detail.outputS3Uri`	chaîne	Préfixe de sortie du compartiment S3 dans lequel les résultats de la tâche d'importation DICOM seront téléchargés.

Événements liés à une série d'images

Image Set Created

État - **`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

État - **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

État - **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

État - **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

État - **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

État - **`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

État - **`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

État - **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

État - **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

État - **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"
```

```
    }
}
```

Événements liés aux ensembles d'images : descriptions des métadonnées

Nom	Type	Description
`version`	chaîne	Version du schéma d' EventBridge événements.
`id`	chaîne	L'UUID version 4 généré pour chaque événement.
`detail-type`	chaîne	Type d'événement envoyé.
`source`	chaîne	Identifie le service qui a généré l'événement.
`account`	chaîne	L'ID de compte AWS à 12 chiffres du propriétaire du magasin de données.
`time`	chaîne	Heure à laquelle l'événement s'est produit.
`region`	chaîne	Identifie la AWS région du magasin de données.
`resources`	tableau (chaîne)	Un tableau JSON qui contient l'ARN du jeu d'images.
`detail`	objet	Un objet JSON qui contient des informations sur l'événement.
`detail.imagingVersion`	chaîne	L'ID de version qui suit les modifications apportées au HealthImaging schéma détaillé des événements.

Nom	Type	Description
`detail.datastoreId`	chaîne	ID du magasin de données qui a généré l'événement de changement de statut.
`detail.imagesetId`	chaîne	L'ID du jeu d'images associé à l'événement de changement de statut.
`detail.imageSetState`	chaîne	État actuel de la série d'images.
`detail.imageSetWorkflowStatus`	chaîne	L'image actuelle définit l'état du flux de travail.

Sécurité dans AWS HealthImaging

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez de centres de données et d'architectures réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cela comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS Cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de AWS conformité Programmes de de conformité. Pour en savoir plus sur les programmes de conformité qui s'appliquent à AWS HealthImaging, voir AWS Services concernés par programme de conformitéAWS .
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris de la sensibilité de vos données, des exigences de votre entreprise, ainsi que de la législation et de la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de son utilisation HealthImaging. Les rubriques suivantes expliquent comment procéder à la configuration HealthImaging pour atteindre vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui vous aident à surveiller et à sécuriser vos HealthImaging ressources.

Rubriques

Protection des données dans AWS HealthImaging

Le modèle de responsabilité AWS partagée de s'applique à la protection des données dans AWS HealthImaging. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée AWS et RGPD (Règlement général sur la protection des données) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- Utilisez le protocole SSL/TLS pour communiquer avec les ressources. AWS Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés par la norme FIPS 140-2 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS (Federal Information Processing Standard) disponibles, consultez Federal Information Processing Standard (FIPS) 140-2 (Normes de traitement de l'information fédérale).

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Name (Nom). Cela inclut lorsque vous travaillez avec HealthImaging ou d'autres Services AWS utilisateurs de la console, de l'API ou AWS des SDK. AWS CLI Toutes les données que vous

entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Rubriques

Chiffrement des données

Avec AWS HealthImaging, vous pouvez ajouter une couche de sécurité à vos données inactives dans le cloud, en fournissant des fonctionnalités de chiffrement évolutives et efficaces. Il s'agit des licences suivantes :

- Fonctionnalités de chiffrement des données au repos disponibles dans la plupart des AWS services
- Des options flexibles de gestion des clés AWS Key Management Service, notamment, avec lesquelles vous pouvez choisir de AWS gérer les clés de chiffrement ou de garder le contrôle total de vos propres clés.
- AWS clés AWS KMS de chiffrement détenues
- Files d'attente de messages chiffrées pour la transmission de données sensibles à l'aide du chiffrement côté serveur (SSE) pour Amazon SQS

En outre, AWS fournit des API vous permettant d'intégrer le chiffrement et la protection des données à tous les services que vous développez ou déployez dans un AWS environnement.

Chiffrement au repos

HealthImaging fournit un chiffrement par défaut pour protéger les données sensibles des clients au repos à l'aide d'une clé appartenant au service AWS KMS .

Chiffrement en transit

HealthImaging utilise le protocole TLS 1.2 pour chiffrer les données en transit via le point de terminaison public et via les services principaux.

Gestion des clés

AWS KMS les clés (clés KMS) constituent la principale ressource de AWS Key Management Service. Vous pouvez également générer des clés de données à utiliser en dehors de AWS KMS.

AWS clé KMS détenue

HealthImaging utilise ces clés par défaut pour chiffrer automatiquement les informations potentiellement sensibles telles que les données personnelles identifiables ou les données de santé privées (PHI) au repos. AWS les clés KMS que vous détenez ne sont pas stockées dans votre compte. Elles font partie d'un ensemble de clés KMS qui AWS possède et gère pour une utilisation dans plusieurs AWS comptes. AWS les services peuvent utiliser AWS des clés KMS détenues pour protéger vos données. Vous ne pouvez pas afficher, gérer, utiliser AWS les clés KMS que vous possédez ou auditer leur utilisation. Cependant, vous n'avez pas besoin de travailler ou de modifier de programme pour protéger les clés qui chiffrent vos données.

Aucuns frais mensuels ni frais d'utilisation ne vous sont facturés si vous utilisez AWS des clés KMS que vous possédez, et elles ne sont pas prises en compte dans les AWS KMS quotas de votre compte. Pour plus d'informations, consultez les clés détenues par AWS dans le manuel du AWS Key Management Service développeur.

Clés KMS gérées par le client

HealthImaging prend en charge l'utilisation d'une clé KMS symétrique gérée par le client que vous créez, détenez et gérez pour ajouter une deuxième couche de chiffrement par rapport au chiffrement que vous AWS possédez déjà. Étant donné que vous avez le contrôle total de cette couche de chiffrement, vous pouvez effectuer les tâches suivantes :

- Établir et maintenir des politiques clés, des politiques IAM et des subventions
- Rotation des matériaux de chiffrement de clé
- Activation et désactivation des stratégies de clé
- Ajout de balises
- Création d'alias de clé
- Planification des clés pour la suppression

Vous pouvez également l' CloudTrail utiliser pour suivre les demandes HealthImaging envoyées AWS KMS en votre nom. Des AWS KMS frais supplémentaires s'appliquent. Pour plus d'informations,

consultez Clés gérées par le client dans le Guide du développeur AWS Key Management Service (langue française non garantie).

Création d'une clé gérée par le client

Vous pouvez créer une clé symétrique gérée par le client à l'aide des API AWS Management Console ou des AWS KMS API. Pour plus d'informations, consultez la section Création de clés KMS de chiffrement symétriques dans le manuel du AWS Key Management Service développeur.

Les politiques de clés contrôlent l'accès à votre clé gérée par le client. Chaque clé gérée par le client doit avoir exactement une stratégie de clé, qui contient des instructions qui déterminent les personnes pouvant utiliser la clé et comment elles peuvent l'utiliser. Lorsque vous créez votre clé gérée par le client, vous pouvez spécifier une stratégie de clé. Pour plus d'informations, consultez Gestion de l'accès aux clés gérées par le client dans le Guide du développeur AWS Key Management Service .

Pour utiliser votre clé gérée par le client avec vos HealthImaging ressources, les CreateGrant opérations kms : doivent être autorisées dans la politique des clés. Cela ajoute une subvention à une clé gérée par le client qui contrôle l'accès à une clé KMS spécifiée, ce qui donne à un utilisateur l'accès aux opérations de subvention HealthImaging requises. Pour plus d'informations, consultez la section Subventions AWS KMS dans le guide du AWS Key Management Service développeur.

Pour utiliser votre clé KMS gérée par le client avec vos HealthImaging ressources, les opérations d'API suivantes doivent être autorisées dans la politique des clés :

- `kms:DescribeKey`fournit les informations clés gérées par le client nécessaires à la validation de la clé. Cela est obligatoire pour toutes les opérations.
- `kms:GenerateDataKey`fournit un accès aux ressources de chiffrement au repos pour toutes les opérations d'écriture.
- `kms:Decrypt`permet d'accéder aux opérations de lecture ou de recherche de ressources chiffrées.
- `kms:ReEncrypt*`permet d'accéder à des ressources de rechiffrement.

Voici un exemple de déclaration de politique qui permet à un utilisateur de créer et d'interagir avec un magasin de HealthImaging données chiffré par cette clé :

```
{
```

```
    "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"
        }
    }
}
```

Autorisations IAM requises pour utiliser une clé KMS gérée par le client

Lors de la création d'un magasin de données dont le AWS KMS chiffrement est activé à l'aide d'une clé KMS gérée par le client, des autorisations sont requises pour la politique de clé et la politique IAM pour l'utilisateur ou le rôle qui crée le magasin de HealthImaging données.

Pour plus d'informations sur les politiques clés, consultez la section Activation des politiques IAM dans le Guide du AWS Key Management Service développeur.

L'utilisateur IAM, le rôle IAM ou le AWS compte qui crée vos référentiels doit disposer des autorisations pour`kms:CreateGrant`,,`kms:GenerateDataKey`, et `kms:RetireGrant` `kms:Decrypt``kms:ReEncrypt*`, ainsi que des autorisations nécessaires pour AWS. HealthImaging

Comment HealthImaging utilise les subventions dans AWS KMS

HealthImaging nécessite une autorisation pour utiliser votre clé KMS gérée par le client. Lorsque vous créez un magasin de données chiffré à l'aide d'une clé KMS gérée par le client, vous HealthImaging créez une subvention en votre nom en envoyant une CreateGrantdemande à AWS KMS. Les subventions AWS KMS sont utilisées pour donner HealthImaging accès à une clé KMS dans un compte client.

Les subventions HealthImaging créées en votre nom ne doivent pas être révoquées ou retirées. Si vous révoquez ou retirez l' HealthImaging autorisation d'utiliser les AWS KMS clés de votre compte, vous HealthImaging ne pouvez pas accéder à ces données, chiffrer les nouvelles ressources d'imagerie envoyées au magasin de données ou les déchiffrer lorsqu'elles sont extraites. Lorsque vous révoquez ou retirez une subvention pour HealthImaging, le changement intervient immédiatement. Pour révoquer les droits d'accès, vous devez supprimer le magasin de données plutôt que de révoquer l'autorisation. Lorsqu'un magasin de données est supprimé, les HealthImaging subventions sont annulées en votre nom.

Surveillance de vos clés de chiffrement pour HealthImaging

Vous pouvez l'utiliser CloudTrail pour suivre les demandes HealthImaging envoyées en votre AWS KMS nom lorsque vous utilisez une clé KMS gérée par le client. Les entrées du CloudTrail journal apparaissent `medical-imaging.amazonaws.com` dans le `userAgent` champ pour distinguer clairement les demandes effectuées par HealthImaging.

Les exemples suivants sont CloudTrail des événements pour`CreateGrant`, `GenerateDataKeyDecrypt`, et `DescribeKey` pour surveiller les AWS KMS opérations appelées HealthImaging pour accéder aux données chiffrées par votre clé gérée par le client.

Ce qui suit montre comment utiliser `CreateGrant` pour autoriser l'accès HealthImaging à une clé KMS fournie par le client, ce qui HealthImaging permet d'utiliser cette clé KMS pour chiffrer toutes les données client au repos.

Les utilisateurs ne sont pas tenus de créer leurs propres subventions. HealthImaging crée une subvention en votre nom en envoyant une `CreateGrant` demande à AWS KMS. Les subventions AWS KMS sont utilisées pour donner HealthImaging accès à une AWS KMS clé dans un compte client.

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

Les exemples suivants montrent comment s'`GenerateDataKey`assurer que l'utilisateur dispose des autorisations nécessaires pour chiffrer les données avant de les stocker.

```
{
    "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'exemple suivant montre comment l'opération HealthImaging appelle l'`Decrypt`opération pour utiliser la clé de données cryptée stockée afin d'accéder aux données cryptées.

```
{
    "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'exemple suivant montre comment HealthImaging utiliser l'`DescribeKey`opération pour vérifier si la AWS KMS clé détenue par le AWS KMS client est dans un état utilisable et pour aider l'utilisateur à résoudre les problèmes si elle n'est pas fonctionnelle.

```
{
    "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"
}
```


En savoir plus

Les ressources suivantes fournissent des informations supplémentaires sur le chiffrement des données au repos et se trouvent dans le manuel du AWS Key Management Service développeur.

- AWS KMS concepts
- Bonnes pratiques en matière de sécurité pour AWS KMS

Confidentialité du trafic réseau

Le trafic est protégé à la fois entre HealthImaging les applications sur site et entre HealthImaging et Amazon S3. Le trafic entre HealthImaging et AWS Key Management Service utilise le protocole HTTPS par défaut.

- AWS HealthImaging est un service régional disponible dans les régions de l'est des États-Unis (Virginie du Nord), de l'ouest des États-Unis (Oregon), de l'Europe (Irlande) et de l'Asie-Pacifique (Sydney).

- Pour le trafic entre les compartiments Amazon S3 HealthImaging et Amazon S3, Transport Layer Security (TLS) chiffre les objets en transit entre et Amazon HealthImaging S3, HealthImaging et entre les applications clients qui y accèdent, vous devez autoriser uniquement les connexions chiffrées via HTTPS (TLS) conformément aux politiques IAM du `aws:SecureTransport` `condition`compartiment Amazon S3. Bien qu'il utilise HealthImaging actuellement le point de terminaison public pour accéder aux données contenues dans les compartiments Amazon S3, cela ne signifie pas que les données transitent par l'Internet public. Tout le trafic entre Amazon S3 HealthImaging et Amazon S3 est acheminé sur le AWS réseau et est chiffré à l'aide du protocole TLS.

Identity and Access Management pour AWS HealthImaging

AWS Identity and Access Management (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser HealthImaging les ressources. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (IAM) varie en fonction du travail que vous effectuez. HealthImaging

Utilisateur du service : si vous utilisez le HealthImaging service pour effectuer votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de nouvelles HealthImaging fonctionnalités pour effectuer votre

travail, vous aurez peut-être besoin d'autorisations supplémentaires. En comprenant bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne pouvez pas accéder à une fonctionnalité dans HealthImaging, consultezRésolution des problèmes liés à HealthImaging l'identité et à l'accès AWS.

Administrateur du service — Si vous êtes responsable des HealthImaging ressources de votre entreprise, vous avez probablement un accès complet à HealthImaging. C'est à vous de déterminer les HealthImaging fonctionnalités et les ressources auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la manière dont votre entreprise peut utiliser IAM avec HealthImaging, voirComment AWS HealthImaging fonctionne avec IAM.

Administrateur IAM : si vous êtes administrateur IAM, vous souhaiterez peut-être en savoir plus sur la manière dont vous pouvez rédiger des politiques pour gérer l'accès à. HealthImaging Pour consulter des exemples de politiques HealthImaging basées sur l'identité que vous pouvez utiliser dans IAM, consultez. Exemples de politiques basées sur l'identité pour AWS HealthImaging

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié (connecté à AWS) en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en AWS tant qu'identité fédérée en utilisant les informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS l'aide de la fédération, vous assumez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter au portail AWS Management Console ou au portail AWS d'accès. Pour plus d'informations sur la connexion à AWS, consultez Comment vous connecter à votre compte Compte AWS dans le guide de Connexion à AWS l'utilisateur.

Si vous y accédez AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes à l'aide

de vos informations d'identification. Si vous n'utilisez pas d' AWS outils, vous devez signer vous-même les demandes. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer vous-même les demandes, consultez la section Signature des demandes AWS d'API dans le guide de l'utilisateur IAM.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, il vous AWS recommande d'utiliser l'authentification multifactorielle (MFA) pour renforcer la sécurité de votre compte. Pour en savoir plus, consultez Authentification multifactorielle dans le Guide de l'utilisateur AWS IAM Identity Center et Utilisation de l'authentification multifactorielle (MFA) dans l'interface AWS dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une identité de connexion unique qui donne un accès complet à toutes Services AWS les ressources du compte. Cette identité est appelée utilisateur Compte AWS root et est accessible en vous connectant avec l'adresse e-mail et le mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur racine et utilisez-les pour effectuer les tâches que seul l'utilisateur racine peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur root, consultez Tâches nécessitant des informations d'identification d'utilisateur root dans le Guide de l'utilisateur IAM.

Identité fédérée

La meilleure pratique consiste à obliger les utilisateurs humains, y compris ceux qui ont besoin d'un accès administrateur, à utiliser la fédération avec un fournisseur d'identité pour accéder à l'aide Services AWS d'informations d'identification temporaires.

Une identité fédérée est un utilisateur de l'annuaire des utilisateurs de votre entreprise, d'un fournisseur d'identité Web AWS Directory Service, du répertoire Identity Center ou de tout utilisateur qui y accède à l'aide des informations d'identification fournies Services AWS par le biais d'une source d'identité. Lorsque des identités fédérées y accèdent Comptes AWS, elles assument des rôles, qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Vous pouvez créer des utilisateurs et des groupes dans IAM Identity Center, ou vous pouvez vous connecter et synchroniser avec un ensemble d'utilisateurs et de groupes dans votre propre

source d'identité afin de les utiliser dans toutes vos applications Comptes AWS et applications. Pour obtenir des informations sur IAM Identity Center, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité au sein de vous Compte AWS qui possède des autorisations spécifiques pour une seule personne ou application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme tels que les clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons de faire pivoter les clés d'accès. Pour plus d'informations, consultez Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification dans le Guide de l'utilisateur IAM.

Un groupe IAM est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez avoir un groupe nommé IAMAdmins et accorder à ce groupe les autorisations d'administrer des ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour en savoir plus, consultez Quand créer un utilisateur IAM (au lieu d'un rôle) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité au sein de vous Compte AWS dotée d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Vous pouvez assumer temporairement un rôle IAM dans le en AWS Management Console changeant de rôle. Vous pouvez assumer un rôle en appelant une opération d' AWS API AWS CLI ou en utilisant une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Utilisation de rôles IAM dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré – Pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie,

l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, consultez la rubrique Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .

- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, dans certains Services AWS cas, vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme proxy). Pour connaître la différence entre les rôles et les politiques basées sur les ressources pour l'accès entre comptes, consultez la section Accès aux ressources entre comptes dans IAM dans le guide de l'utilisateur IAM.
- Accès multiservices — Certains Services AWS utilisent des fonctionnalités dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, un rôle de service ou un rôle lié au service.
  - Sessions d'accès direct (FAS) : lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal appelant et Service AWS, associées Service AWS à la demande, pour adresser des demandes aux services en aval. Les demandes FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez Sessions de transmission d'accès.
  - Rôle de service : il s'agit d'un rôle IAM attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer une fonction du service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.

- Rôle lié à un service — Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés au service apparaissent dans votre Compte AWS fichier et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

- Applications exécutées sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer les informations d'identification temporaires pour les applications qui s'exécutent sur une instance EC2 et qui envoient des demandes d'API. AWS CLI AWS Cette solution est préférable au stockage des clés d'accès au sein de l'instance EC2. Pour attribuer un AWS rôle à une instance EC2 et le mettre à la disposition de toutes ses applications, vous devez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes qui s'exécutent sur l'instance EC2 d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez Utilisation d'un rôle IAM pour accorder des autorisations à des applications s'exécutant sur des instances Amazon EC2 dans le Guide de l'utilisateur IAM.

Pour savoir dans quel cas utiliser des rôles ou des utilisateurs IAM, consultez Quand créer un rôle IAM (au lieu d'un utilisateur) dans le Guide de l'utilisateur IAM.

Gestion des accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique est un objet AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit leurs autorisations. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur root ou session de rôle) fait une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations sur la structure et le contenu des documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d'une politique qui autorise l'action `iam:GetRole`. Un utilisateur appliquant cette politique peut obtenir des informations sur le rôle à partir de AWS Management Console AWS CLI, de ou de l' AWS API.

Politiques basées sur l'identité

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Création de politiques IAM dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez associer à plusieurs utilisateurs, groupes et rôles au sein de votre Compte AWS. Les politiques gérées incluent les politiques AWS gérées et les politiques gérées par le client. Pour découvrir comment choisir entre une politique gérée et une politique en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Des politiques basées sur les ressources sont, par exemple, les politiques de confiance de rôle IAM et des politiques de compartiment. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d’accès (ACL)

Les listes de contrôle d’accès (ACL) vérifie quels principals (membres de compte, utilisateurs ou rôles) ont l’autorisation d’accéder à une ressource. Les listes de contrôle d’accès sont similaires aux politiques basées sur les ressources, bien qu’elles n’utilisent pas le format de document de politique JSON.

Amazon S3 et Amazon VPC sont des exemples de services qui prennent en charge les ACL. AWS WAF Pour en savoir plus sur les listes de contrôle d’accès, consultez Vue d’ensemble des listes de contrôle d’accès (ACL) dans le Guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge d'autres types de politiques moins courants. Ces types de politiques peuvent définir le nombre maximum d’autorisations qui vous sont accordées par des types de politiques plus courants.

- Limite d’autorisations : une limite d’autorisations est une fonctionnalité avancée dans laquelle vous définissez le nombre maximal d’autorisations qu’une politique basée sur l’identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d’autorisations pour une entité. Les autorisations en résultant représentent la combinaison des politiques basées sur l’identité d’une entité et de ses limites d’autorisation. Les politiques basées sur les ressources qui spécifient l’utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d’autorisations. Un refus explicite dans l’une de ces politiques remplace l’autorisation. Pour plus d’informations sur les limites d’autorisations, consultez Limites d’autorisations pour des entités IAM dans le Guide de l’utilisateur IAM.
- Politiques de contrôle des services (SCP) — Les SCP sont des politiques JSON qui spécifient les autorisations maximales pour une organisation ou une unité organisationnelle (UO) dans. AWS Organizations AWS Organizations est un service permettant de regrouper et de gérer de manière centralisée Comptes AWS les multiples propriétés de votre entreprise. Si vous activez toutes les fonctionnalités d’une organisation, vous pouvez appliquer les politiques de contrôle des services (SCP) à l’un ou à l’ensemble de vos comptes. Le SCP limite les autorisations pour les entités figurant dans les comptes des membres, y compris chacune Utilisateur racine d'un compte AWS d'entre elles. Pour plus d’informations sur les organisations et les SCP, consultez Fonctionnement des SCP dans le Guide de l’utilisateur AWS Organizations .
- Politiques de séance : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de séance en résultant sont une combinaison des politiques

basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations, consultez politiques de séance dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Comment AWS HealthImaging fonctionne avec IAM

Avant d'utiliser IAM pour gérer l'accès à HealthImaging, découvrez les fonctionnalités IAM disponibles. HealthImaging

Fonctionnalités IAM que vous pouvez utiliser avec AWS HealthImaging

Fonction IAM	HealthImaging soutien
Politiques basées sur l'identité	Oui
Politiques basées sur les ressources	Non
Actions de politique	Oui
Ressources de politique	Oui
Clés de condition de politique (spécifiques au service)	Oui
ACL	Non
ABAC (identifications dans les politiques)	Partielle
Informations d'identification temporaires	Oui
Autorisations de principal	Oui

Fonction IAM	HealthImaging soutien
Fonctions de service	Oui
Rôles liés à un service	Non

Pour obtenir une vue d'ensemble de la façon dont HealthImaging les autres AWS services fonctionnent avec la plupart des fonctionnalités IAM, consultez la section AWS Services compatibles avec IAM dans le Guide de l'utilisateur IAM.

Politiques basées sur l'identité pour HealthImaging

Prend en charge les politiques basées sur l’identité	Oui

Les politiques basées sur l’identité sont des documents de politique d’autorisations JSON que vous pouvez attacher à une identité telle qu’un utilisateur, un groupe d’utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d’actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l’identité, consultez Création de politiques IAM dans le Guide de l’utilisateur IAM.

Avec les politiques IAM basées sur l’identité, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. Vous ne pouvez pas spécifier le principal dans une politique basée sur une identité car celle-ci s’applique à l’utilisateur ou au rôle auquel elle est attachée. Pour découvrir tous les éléments que vous utilisez dans une politique JSON, consultez Références des éléments de politique JSON IAM dans le Guide de l’utilisateur IAM.

Exemples de politiques basées sur l'identité pour HealthImaging

Pour consulter des exemples de politiques HealthImaging basées sur l'identité, consultez. Exemples de politiques basées sur l'identité pour AWS HealthImaging

Politiques basées sur les ressources au sein de HealthImaging

Prend en charge les politiques basées sur les ressources	Non

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Des politiques basées sur les ressources sont, par exemple, les politiques de confiance de rôle IAM et des politiques de compartiment. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Pour permettre un accès intercompte, vous pouvez spécifier un compte entier ou des entités IAM dans un autre compte en tant que principal dans une politique basée sur les ressources. L'ajout d'un principal entre comptes à une politique basée sur les ressources ne représente qu'une partie de l'instauration de la relation d'approbation. Lorsque le principal et la ressource sont différents Comptes AWS, un administrateur IAM du compte sécurisé doit également accorder à l'entité principale (utilisateur ou rôle) l'autorisation d'accéder à la ressource. Pour ce faire, il attache une politique basée sur une identité à l'entité. Toutefois, si une politique basée sur des ressources accorde l'accès à un principal dans le même compte, aucune autre politique basée sur l'identité n'est requise. Pour plus d'informations, consultez la section Accès aux ressources entre comptes dans IAM dans le guide de l'utilisateur d'IAM.

Actions politiques pour HealthImaging

Prend en charge les actions de politique	Oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Action` d'une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une politique. Les actions de stratégie portent généralement le même nom

que l'opération AWS d'API associée. Il existe quelques exceptions, telles que les actions avec autorisations uniquement qui n'ont pas d'opération API correspondante. Certaines opérations nécessitent également plusieurs actions dans une politique. Ces actions supplémentaires sont nommées actions dépendantes.

Intégration d'actions dans une stratégie afin d'accorder l'autorisation d'exécuter les opérations associées.

Pour consulter la liste des HealthImaging actions, consultez la section Actions définies par AWS HealthImaging dans le Service Authorization Reference.

Les actions de politique en HealthImaging cours utilisent le préfixe suivant avant l'action :

```
AWS
```

Pour indiquer plusieurs actions dans une seule déclaration, séparez-les par des virgules.

```
"Action": [
      "AWS:action1",
      "AWS:action2"
         ]
```

Pour consulter des exemples de politiques HealthImaging basées sur l'identité, consultez. Exemples de politiques basées sur l'identité pour AWS HealthImaging

Ressources politiques pour HealthImaging

Prend en charge les ressources de politique	Oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets auxquels l'action s'applique. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de définir

une ressource à l'aide de son Amazon Resource Name (ARN). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Pour consulter la liste des types de HealthImaging ressources et leurs ARN, consultez la section Types de ressources définis par AWS HealthImaging dans le Service Authorization Reference. Pour savoir avec quelles actions et ressources vous pouvez utiliser un ARN, consultez la section Actions définies par AWS HealthImaging.

Pour consulter des exemples de politiques HealthImaging basées sur l'identité, consultez. Exemples de politiques basées sur l'identité pour AWS HealthImaging

Clés de conditions de politique pour HealthImaging

Prend en charge les clés de condition de politique spécifiques au service	Oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` (ou le bloc `Condition`) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément `Condition` est facultatif. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande.

Si vous spécifiez plusieurs éléments `Condition` dans une instruction, ou plusieurs clés dans un seul élément `Condition`, AWS les évalue à l'aide d'une opération `AND` logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une `OR`

opération logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez accorder à un utilisateur IAM l'autorisation d'accéder à une ressource uniquement si elle est balisée avec son nom d'utilisateur IAM. Pour plus d'informations, consultez Éléments d'une politique IAM : variables et identifications dans le Guide de l'utilisateur IAM.

AWS prend en charge les clés de condition globales et les clés de condition spécifiques au service. Pour voir toutes les clés de condition AWS globales, voir les clés de contexte de condition AWS globales dans le guide de l'utilisateur IAM.

Pour consulter la liste des clés de HealthImaging condition, consultez la section Clés de condition pour AWS HealthImaging dans la référence d'autorisation de service. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez Actions définies par AWS HealthImaging.

Pour consulter des exemples de politiques HealthImaging basées sur l'identité, consultez. Exemples de politiques basées sur l'identité pour AWS HealthImaging

ACL dans HealthImaging

Prend en charge les listes ACL	Non

Les listes de contrôle d'accès (ACL) vérifient quels principaux (membres de compte, utilisateurs ou rôles) ont l'autorisation d'accéder à une ressource. Les listes de contrôle d'accès sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

RBAC avec HealthImaging

Supporte le RBAC	Oui

Le modèle d'autorisation traditionnel utilisé dans IAM est appelé contrôle d'accès basé sur les rôles (RBAC). Le RBAC définit les autorisations en fonction de la fonction professionnelle d'une personne, connue en dehors du nom de AWS rôle. Pour plus d'informations, voir Comparaison de l'ABAC au modèle RBAC traditionnel dans le guide de l'utilisateur IAM.

ABAC avec HealthImaging

Prise en charge d'ABAC (identifications dans les politiques)	Partielle

> **Warning**
>
> L'ABAC n'est pas appliqué via l'action de l'`SearchImageSets`API. Toute personne ayant accès à l'`SearchImageSets`action peut accéder à toutes les métadonnées des ensembles d'images d'un magasin de données.

> **Note**
>
> Les ensembles d'images sont une ressource dérivée des banques de données. Pour utiliser ABAC, un ensemble d'images doit avoir le même tag qu'un magasin de données. Pour plus d'informations, consultez Marquage des ressources avec AWS HealthImaging.

Le contrôle d'accès par attributs (ABAC) est une stratégie d'autorisation qui définit des autorisations en fonction des attributs. Dans AWS, ces attributs sont appelés balises. Vous pouvez associer des balises aux entités IAM (utilisateurs ou rôles) et à de nombreuses AWS ressources. L'étiquetage des entités et des ressources est la première étape d'ABAC. Vous concevez ensuite des politiques ABAC pour autoriser des opérations quand l'identification du principal correspond à celle de la ressource à laquelle il tente d'accéder.

L'ABAC est utile dans les environnements qui connaissent une croissance rapide et pour les cas où la gestion des politiques devient fastidieuse.

Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'élément de condition d'une politique utilisant les clés de condition `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* ou `aws:TagKeys`.

Si un service prend en charge les trois clés de condition pour tous les types de ressources, alors la valeur pour ce service est Oui. Si un service prend en charge les trois clés de condition pour certains types de ressources uniquement, la valeur est Partielle.

Pour plus d'informations sur l'ABAC, consultez Qu'est-ce que le contrôle d'accès basé sur les attributs (ABAC) ? dans le Guide de l'utilisateur IAM. Pour accéder à un didacticiel décrivant les étapes de configuration de l'ABAC, consultez Utilisation du contrôle d'accès par attributs (ABAC) dans le Guide de l'utilisateur IAM.

Utilisation d'informations d'identification temporaires avec HealthImaging

Prend en charge les informations d'identification temporaires	Oui

Certains Services AWS ne fonctionnent pas lorsque vous vous connectez à l'aide d'informations d'identification temporaires. Pour plus d'informations, y compris celles qui Services AWS fonctionnent avec des informations d'identification temporaires, consultez Services AWS la section relative à l'utilisation d'IAM dans le guide de l'utilisateur d'IAM.

Vous utilisez des informations d'identification temporaires si vous vous connectez à l' AWS Management Console aide d'une méthode autre qu'un nom d'utilisateur et un mot de passe. Par exemple, lorsque vous accédez à AWS l'aide du lien d'authentification unique (SSO) de votre entreprise, ce processus crée automatiquement des informations d'identification temporaires. Vous créez également automatiquement des informations d'identification temporaires lorsque vous vous connectez à la console en tant qu'utilisateur, puis changez de rôle. Pour plus d'informations sur le changement de rôle, consultez Changement de rôle (console) dans le Guide de l'utilisateur IAM.

Vous pouvez créer manuellement des informations d'identification temporaires à l'aide de l' AWS API AWS CLI or. Vous pouvez ensuite utiliser ces informations d'identification temporaires pour y accéder AWS. AWS recommande de générer dynamiquement des informations d'identification temporaires au lieu d'utiliser des clés d'accès à long terme. Pour plus d'informations, consultez Informations d'identification de sécurité temporaires dans IAM.

Autorisations principales interservices pour HealthImaging

Prend en charge les sessions d'accès direct (FAS)	Oui

Lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Les politiques accordent des autorisations au principal. Lorsque vous

utilisez certains services, vous pouvez effectuer une action qui déclenche une autre action dans un autre service. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour savoir si une action nécessite des actions dépendantes supplémentaires dans une politique, consultez la section Actions, ressources et clés de condition pour AWS HealthImaging dans la référence d'autorisation de service.

Fonctions du service pour HealthImaging

Prend en charge les fonctions du service	Oui

Une fonction de service est un rôle IAM qu'un service endosse pour accomplir des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer une fonction du service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.

> **Warning**
>
> La modification des autorisations associées à un rôle de service peut perturber HealthImaging les fonctionnalités. Modifiez les rôles de service uniquement lorsque HealthImaging vous êtes invité à le faire.

Rôles liés à un service pour HealthImaging

Prend en charge les rôles liés à un service	Non

Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés au service apparaissent dans votre Compte AWS fichier et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

Pour plus d'informations sur la création ou la gestion des rôles liés à un service, consultez Services AWS qui fonctionnent avec IAM. Recherchez un service dans le tableau qui inclut un `Yes` dans la colonne Rôle lié à un service. Choisissez le lien Oui pour consulter la documentation du rôle lié à ce service.

Exemples de politiques basées sur l'identité pour AWS HealthImaging

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou à modifier HealthImaging des ressources. Ils ne peuvent pas non plus effectuer de tâches à l'aide de l'API AWS Management Console, AWS Command Line Interface (AWS CLI) ou de AWS l'API. Pour octroyer aux utilisateurs des autorisations d’effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L’administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Pour apprendre à créer une politique basée sur l’identité IAM à l’aide de ces exemples de documents de politique JSON, consultez Création de politiques dans l’onglet JSON dans le Guide de l’utilisateur IAM.

Pour plus de détails sur les actions et les types de ressources définis par Awesome, y compris le format des ARN pour chacun des types de ressources, voir Actions, ressources et clés de condition pour AWS Awesome dans la référence d'autorisation du service.

Rubriques

Bonnes pratiques en matière de politiques

Les politiques basées sur l'identité déterminent si quelqu'un peut créer, accéder ou supprimer HealthImaging des ressources dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l’identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d’informations, consultez politiques gérées par AWS ou politiques gérées par AWS pour les activités professionnelles dans le Guide de l’utilisateur IAM.

- Accorder les autorisations de moindre privilège : lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation de IAM pour appliquer des autorisations, consultez politiques et autorisations dans IAM dans le Guide de l'utilisateur IAM.
- Utiliser des conditions dans les politiques IAM pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que AWS CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.
- Utilisez IAM Access Analyzer pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : IAM Access Analyzer valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez Validation de politique IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger le MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez Configuration de l'accès aux API protégé par MFA dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.

Utilisation de la console HealthImaging

Pour accéder à la HealthImaging console AWS, vous devez disposer d'un ensemble minimal d'autorisations. Ces autorisations doivent vous permettre de répertorier et d'afficher les détails HealthImaging des ressources de votre Compte AWS. Si vous créez une stratégie basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette stratégie.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l' AWS API. Autorisez plutôt l'accès à uniquement aux actions qui correspondent à l'opération d'API qu'ils tentent d'effectuer.

Pour garantir que les utilisateurs et les rôles peuvent toujours utiliser la HealthImaging console, associez également la politique HealthImaging *ConsoleAccess* ou la politique *ReadOnly* AWS gérée aux entités. Pour plus d'informations, consultez Ajout d'autorisations à un utilisateur dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or 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": "*"
        }
    ]
}
```

AWSpolitiques gérées pour AWS HealthImaging

Une politique gérée par AWS est une politique autonome créée et administrée par AWS. Les politiques gérées par AWS sont conçues pour fournir des autorisations pour de nombreux cas d'utilisation courants afin que vous puissiez commencer à attribuer des autorisations aux utilisateurs, aux groupes et aux rôles.

Gardez à l'esprit que les politiques gérées par AWS peuvent ne pas accorder les autorisations de moindre privilège pour vos cas d'utilisation spécifiques, car elles sont disponibles pour tous les clients AWS. Nous vous recommandons de réduire encore les autorisations en définissant des politiques gérées par le client qui sont propres à vos cas d'utilisation.

Vous ne pouvez pas modifier les autorisations définies dans les stratégies gérées par AWS. Si AWS met à jour les autorisations définies dans une politique gérée par AWS, la mise à jour affecte toutes les identités de principal (utilisateurs, groupes et rôles) auxquelles la politique est associée. AWS est plus susceptible de mettre à jour une politique gérée par AWS lorsqu'un nouveau Service AWS est lancé ou que de nouvelles opérations API deviennent accessibles pour les services existants.

Pour plus d'informations, consultez la rubrique Politiques gérées par AWS dans le Guide de l'utilisateur IAM.

Rubriques

- AWSpolitique gérée : AWSHealthImagingFullAccess
- AWSpolitique gérée : AWSHealthImagingReadOnlyAccess
- HealthImaging mises à jour des politiques AWS gérées

AWSpolitique gérée : AWSHealthImagingFullAccess

Vous pouvez attacher la politique `AWSHealthImagingFullAccess` à vos identités IAM.

Cette politique accorde des autorisations administratives pour toutes les HealthImaging actions.

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

AWSpolitique gérée : AWSHealthImagingReadOnlyAccess

Vous pouvez attacher la politique `AWSHealthImagingReadOnlyAccess` à vos identités IAM.

Cette politique accorde une autorisation en lecture seule à des actions AWS HealthImaging spécifiques.

```
{
    "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 mises à jour des politiques AWS gérées

Consultez les détails des mises à jour des politiques AWS gérées HealthImaging depuis que ce service a commencé à suivre ces modifications. Pour recevoir des alertes automatiques concernant les modifications apportées à cette page, abonnez-vous au flux RSS sur la page des communiqués.

Modification	Description	Date
HealthImaging a commencé à suivre les modifications	HealthImaging a commencé à suivre les modifications apportées AWS à ses politique s gérées.	19 juillet 2023

Résolution des problèmes liés à HealthImaging l'identité et à l'accès AWS

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec HealthImaging IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans HealthImaging

- Je ne suis pas autorisé à effectuer iam : PassRole
- Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes HealthImaging ressources

Je ne suis pas autorisé à effectuer une action dans HealthImaging

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *`my-example-widget`* fictive, mais ne dispose pas des autorisations `AWS:`*`GetWidget`* fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 AWS:GetWidget on resource: my-example-widget
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *`my-example-widget`* à l'aide de l'action `AWS:`*`GetWidget`*.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez un message d'erreur indiquant que vous n'êtes pas autorisé à effectuer l'`iam:PassRole`action, vos politiques doivent être mises à jour pour vous permettre de transmettre un rôle HealthImaging.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, un utilisateur doit disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour effectuer une action dans HealthImaging. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes HealthImaging ressources

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACL), vous pouvez utiliser ces politiques pour donner l'accès à vos ressources.

Pour en savoir plus, consultez les éléments suivants :

- Pour savoir si ces fonctionnalités sont prises HealthImaging en charge, consultezComment AWS HealthImaging fonctionne avec IAM.
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour connaître la différence entre l'utilisation de rôles et de politiques basées sur les ressources pour l'accès entre comptes, consultez la section Accès aux ressources entre comptes dans IAM dans le guide de l'utilisateur d'IAM.

Validation de conformité pour AWS HealthImaging

Des auditeurs tiers évaluent la sécurité et la conformité d'AWS dans HealthImaging le cadre de plusieurs programmes de AWS conformité. Car cela inclut HealthImaging la loi HIPAA.

Pour obtenir la liste des services AWS relevant de programmes de conformité spécifiques, consultez les Services AWS relevant de programmes de conformité. Pour obtenir des renseignements généraux, consultez Programmes de conformité AWS.

Vous pouvez télécharger les rapports de l'audit externe avec AWS Artifact. Pour plus d'informations, consultez Téléchargement des rapports dans AWS Artifact.

Lorsque vous utilisez AWS HealthImaging , votre responsabilité en matière de conformité dépend de la sensibilité de vos données, des objectifs de conformité de votre entreprise et des lois et réglementations applicables. AWSfournit les ressources suivantes pour faciliter la mise en conformité :

- AWSSolutions pour les partenaires — Les guides de déploiement de référence automatisés pour la sécurité et la conformité abordent les considérations architecturales et fournissent des étapes pour déployer des environnements de référence axés sur la sécurité et la conformité sur. AWS
- Livre blanc sur l'architecture pour la sécurité et la conformité HIPAA : le livre blanc décrit comment les entreprises peuvent utiliser AWS pour créer des applications conformes à HIPAA.
- GxP Systems on AWS — Ce livre blanc fournit des informations sur la manière d'AWSaborder la conformité et la sécurité liées au GxP et fournit des conseils sur l'utilisation des AWS services dans le contexte de GxP.
- Ressources de conformité AWS : cet ensemble de manuels et de guides peut s'appliquer à votre secteur et à votre emplacement.
- Évaluation des ressources à l'aide de règles : AWS Config évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- AWS Security Hub : ce service AWS fournit une vue complète de votre état de sécurité au sein d'AWS qui vous permet de vérifier votre conformité aux normes du secteur et aux bonnes pratiques de sécurité.

Sécurité de l'infrastructure dans AWS HealthImaging

En tant que service géré, AWS HealthImaging est protégé par les procédures de sécurité du réseau AWS mondial décrites dans le livre blanc Amazon Web Services : présentation des processus de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder HealthImaging via le réseau. Les clients doivent prendre en charge le protocole TLS (Transport Layer Security) 1.3 ou version ultérieure. Les

clients doivent aussi prendre en charge les suites de chiffrement PFS (Perfect Forward Secrecy) comme Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser le AWS Security Token Service(AWS STS) pour générer des informations de sécurité temporaires afin de signer les demandes.

Création de HealthImaging ressources AWS avec AWS CloudFormation

AWS HealthImaging est intégré à AWS CloudFormation un service qui vous aide à modéliser et à configurer vos AWS ressources afin que vous puissiez passer moins de temps à créer et à gérer vos ressources et votre infrastructure. Vous créez un modèle qui décrit toutes les AWS ressources que vous souhaitez et qui AWS CloudFormation fournit et configure ces ressources pour vous.

Lorsque vous l'utilisezAWS CloudFormation, vous pouvez réutiliser votre modèle pour configurer vos HealthImaging ressources de manière cohérente et répétée. Décrivez vos ressources une seule fois, puis allouez-les autant de fois que vous le souhaitez dans plusieurs Comptes AWS et régions.

HealthImaging et AWS CloudFormation modèles

Pour fournir et configurer des ressources HealthImaging et des services associés, vous devez comprendre les AWS CloudFormationmodèles. Les modèles sont des fichiers texte formatés en JSON ou YAML. Ces modèles décrivent les ressources que vous souhaitez allouer dans vos piles AWS CloudFormation. Si JSON ou YAML ne vous est pas familier, vous pouvez utiliser AWS CloudFormation Designer pour vous aider à démarrer avec des modèles AWS CloudFormation. Pour plus d'informations, consultez Qu'est-ce que AWS CloudFormation Designer ? dans le AWS CloudFormationGuide de l'utilisateur.

AWS HealthImaging prend en charge la création de magasins de données avecAWS CloudFormation. Pour plus d'informations, notamment des exemples de modèles JSON et YAML pour le provisionnement de magasins de HealthImaging données, consultez la référence sur les types de HealthImaging ressources AWS dans le guide de l'AWS CloudFormationutilisateur.

En savoir plus sur AWS CloudFormation

Pour en savoir plus sur AWS CloudFormation, consultez les ressources suivantes :

- AWS CloudFormation
- Guide de l'utilisateur AWS CloudFormation
- Référence API AWS CloudFormation
- Guide de l'utilisateur de l'interface de ligne de commande AWS CloudFormation

AWS HealthImaging et points de terminaison VPC d'interface ()AWS PrivateLink

Vous pouvez établir une connexion privée entre votre VPC et créer un point de AWS HealthImaging terminaison VPC d'interface. Les points de terminaison d'interface sont alimentés par AWS PrivateLinkune technologie que vous pouvez utiliser pour accéder à HealthImaging des API de manière privée sans passerelle Internet, appareil NAT, connexion VPN ou connexion AWS Direct Connect. Les instances de votre VPC n'ont pas besoin d'adresses IP publiques pour communiquer avec HealthImaging les API. Le trafic entre votre VPC et celui qui HealthImaging ne quitte pas le réseau Amazon.

Chaque point de terminaison d'interface est représenté par une ou plusieurs interfaces réseau Elastic dans vos sous-réseaux.

Pour plus d'informations, consultez la section Interface VPC endpoints (AWS PrivateLink) dans le guide de l'utilisateur Amazon VPC.

Rubriques

- Considérations relatives aux points de HealthImaging terminaison VPC
- Création d'un point de terminaison VPC d'interface pour HealthImaging
- Création d'une politique de point de terminaison VPC pour HealthImaging

Considérations relatives aux points de HealthImaging terminaison VPC

Avant de configurer un point de terminaison VPC d'interface pour HealthImaging, assurez-vous de consulter les propriétés et les limites du point de terminaison d'interface dans le guide de l'utilisateur Amazon VPC.

HealthImaging permet d'appeler toutes les AWS HealthImaging actions depuis votre VPC.

Création d'un point de terminaison VPC d'interface pour HealthImaging

Vous pouvez créer un point de terminaison VPC pour le HealthImaging service à l'aide de la console Amazon VPC ou du (). AWS Command Line Interface AWS CLI Pour plus d'informations, consultez Création d'un point de terminaison d'interface dans le Guide de l'utilisateur Amazon VPC.

Créez des points de terminaison VPC pour HealthImaging utiliser les noms de service suivants :

- com.amazonaws. *région .imagerie* médicale
- com.amazonaws. *région*. runtime-medical-imaging
- com.amazonaws. *région*. dicom-medical-imaging

> **Note**
>
> Le DNS privé doit être activé pour être utilisé PrivateLink.

Vous pouvez envoyer des demandes d'API à HealthImaging l'aide de son nom DNS par défaut pour la région, par exemple,`medical-imaging.us-east-1.amazonaws.com`.

Pour plus d'informations, consultez Accès à un service via un point de terminaison d'interface dans le Guide de l'utilisateur Amazon VPC.

Création d'une politique de point de terminaison VPC pour HealthImaging

Vous pouvez associer une politique de point de terminaison à votre point de terminaison VPC qui contrôle l'accès à. HealthImaging La politique spécifie les informations suivantes :

- Le principal qui peut exécuter des actions.
- Les actions qui peuvent être effectuées.
- Les ressources sur lesquelles les actions peuvent être exécutées.

Pour plus d'informations, consultez Contrôle de l'accès aux services avec points de terminaison d'un VPC dans le Guide de l'utilisateur Amazon VPC.

Exemple : politique de point de terminaison VPC pour les actions HealthImaging

Voici un exemple de politique de point de terminaison pour HealthImaging. Lorsqu'elle est attachée à un point de terminaison, cette politique accorde l'accès aux HealthImaging actions pour tous les principaux sur toutes les ressources.

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\":\"*\"}]}"
```

Importation entre comptes pour AWS HealthImaging

Grâce à l'importation entre comptes ou entre régions, vous pouvez importer des données dans votre HealthImaging magasin de données à partir de compartiments Amazon S3 situés dans d'autres régions prises en charge. Vous pouvez importer des données entre des AWS comptes, des comptes appartenant à d'autres AWS Organisations et à partir de sources de données ouvertes telles que Imaging Data Commons (IDC) situées dans le Registre des données ouvertes sur AWS.

HealthImaging les cas d'utilisation de l'importation entre comptes ou entre régions incluent :

- Produits SaaS d'imagerie médicale qui importent des données DICOM depuis les comptes clients

- Grandes entreprises alimentant un magasin de HealthImaging données à partir de nombreux compartiments d'entrée Amazon S3
- Les chercheurs partagent en toute sécurité des données dans le cadre d'études cliniques menées dans plusieurs établissements

Pour utiliser l'importation entre comptes

1. Le propriétaire du compartiment d'entrée (source) Amazon S3 doit accorder au propriétaire du magasin de HealthImaging données `s3:ListBucket` et `s3:GetObject` les autorisations nécessaires.
2. Le propriétaire du magasin de HealthImaging données doit ajouter le compartiment Amazon S3 à son IAM`ImportJobDataAccessRole`. veuillez consulter Création d'un rôle IAM pour l'importation.
3. Le propriétaire du magasin de HealthImaging données doit fournir le `inputOwnerAccountId`compartiment d'entrée Amazon S3 lors du démarrage de la tâche d'importation.

> **Note**
>
> En fournissant le`inputOwnerAccountId`, le propriétaire du magasin de données valide que le compartiment Amazon S3 d'entrée appartient au compte spécifié afin de garantir la conformité aux normes du secteur et d'atténuer les risques de sécurité potentiels.

L'exemple de `startDICOMImportJob` code suivant inclut le `inputOwnerAccountId` paramètre facultatif, qui peut être appliqué à tous, AWS CLI ainsi que les exemples de code du SDK présentés dans la Démarrage d'une tâche d'importation section.

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 "";
}
```

Résilience dans AWS HealthImaging

L'infrastructure mondiale d'AWS est construite autour de zones de disponibilité et de Régions AWS. Les Régions AWSfournissent plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, à débit élevé et à forte redondance. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur les Régions AWS et les zones de disponibilité, consultez Infrastructure mondiale d'AWS.

Outre l'infrastructure AWS mondiale, AWS HealthImaging propose plusieurs fonctionnalités pour répondre à vos besoins en matière de résilience et de sauvegarde des données.

Matériel HealthImaging de référence AWS

Le matériel de référence suivant est disponible pour AWS HealthImaging.

> **Note**
>
> Toutes les HealthImaging actions et tous les types de données se trouvent dans une référence distincte. Pour plus d'informations, consultez le manuel de référence des HealthImaging API AWS.

Rubriques

Support DICOM pour AWS HealthImaging

AWS HealthImaging prend en charge des éléments DICOM et des syntaxes de transfert spécifiques. Familiarisez-vous avec les éléments de données DICOM pris en charge au niveau du patient, de l'étude et de la série, car les clés de HealthImaging métadonnées sont basées sur eux. Avant de commencer une importation, vérifiez que vos données d'imagerie médicale sont conformes aux syntaxes HealthImaging de transfert prises en charge et aux contraintes relatives aux éléments DICOM.

> **Note**
>
> AWS HealthImaging ne prend actuellement pas en charge les images de segmentation binaire ou les données en pixels de séquences d'images d'icônes.

Rubriques

Classes SOP prises en charge

Avec AWS HealthImaging, vous pouvez importer des instances DICOM P10 Service-Object Pair (SOP) codées avec n'importe quel UID de classe SOP, y compris les instances retirées et privées. Tous les attributs privés sont également préservés.

Normalisation des métadonnées

Lorsque vous importez vos données DICOM P10 dans AWS HealthImaging, elles sont transformées en ensembles d'images composés de métadonnées et de cadres d'image (données en pixels). Au cours du processus de transformation, HealthImaging les clés de métadonnées sont générées sur la base d'une version spécifique de la norme DICOM. HealthImaging génère et prend actuellement en charge des clés de métadonnées basées sur le dictionnaire de données DICOM PS3.6 2022b.

AWS HealthImaging prend en charge les éléments de données DICOM suivants au niveau du patient, de l'étude et de la série.

Éléments relatifs au niveau du patient

> **Note**
>
> Pour une description détaillée de chaque élément au niveau du patient, consultez le Registre des éléments de données DICOM.

AWS HealthImaging prend en charge les éléments suivants au niveau du patient :

```
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
```

Éléments du niveau de l'étude

> **Note**
>
> Pour une description détaillée de chaque élément du niveau de l'étude, consultez le Registre des éléments de données DICOM.

AWS HealthImaging prend en charge les éléments de niveau étude suivants :

```
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
```

Éléments au niveau de la série

> **Note**
>
> Pour une description détaillée de chaque élément de niveau série, consultez le registre des éléments de données DICOM.

AWS HealthImaging prend en charge les éléments suivants au niveau de la série :

```
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
```

Syntaxes de transfert prises en charge

AWS HealthImaging importe des instances SOP DICOM P10 codées avec les syntaxes de transfert indiquées dans le tableau suivant. Outre le stockage de l'instance SOP, HealthImaging transcode les images (données de pixels) en HTJ2K pour les instances SOP codées avec les syntaxes de transfert suivantes :

Syntaxe de transfert UID	Nom de la syntaxe de transfert
1.2.840.10008,1.2	Endian VR implicite : syntaxe de transfert par défaut pour DICOM
1.2.840.10008.1.2.1	Little Endian en réalité virtuelle explicite
1.2.840.10008.1.2.1.99	Little Endian explicite en réalité virtuelle dégonflé
1.2.840.10008.1.2.2	Big Endian en réalité virtuelle explicite
1.2.840.10008.1.2.4.50	Ligne de base JPEG (processus 1) : syntaxe de transfert par défaut pour la compression d'image JPEG 8 bits avec perte
1.2.840.10008.1.2.4.51	Ligne de base JPEG (processus 2 et 4) : syntaxe de transfert par défaut pour la

Syntaxe de transfert UID	Nom de la syntaxe de transfert
	compression d'image JPEG 12 bits avec perte (processus 4 uniquement)
1.2.840.10008.1.2.4.57	JPEG sans perte non hiérarchique (processus 14)
1.2.840.10008.1.2.4.70	Prédiction JPEG sans perte, non hiérarchique, du premier ordre (processus 14 [Valeur de sélection 1]) : syntaxe de transfert par défaut pour la compression d'images JPEG sans perte
1.2.840.10008.1.2.4.80	Compression d'image sans perte JPEG-LS
1.2.840.10008.1.2.4.81	Compression d'image JPEG-LS avec perte (quasi sans perte)
1.2.840.10008.1.2.4.90	Compression d'image JPEG 2000 (sans perte uniquement)
1.2.840.10008.1.2.4.91	Compression d'image JPEG 2000
1.2.840.10008.1.2.4.201	Compression d'image JPEG 2000 à haut débit (sans perte uniquement)
1.2.840.10008.1.2.4.202	JPEG 2000 à haut débit avec options RPCL, compression d'image (sans perte uniquement)
1.2.840.10008.1.2.4.203	Compression d'image JPEG 2000 à haut débit
1.2.840.10008.1.2.5	RLE sans perte

Contraintes relatives aux éléments DICOM

Lorsque vous importez vos données d'imagerie médicale dans AWS HealthImaging, des contraintes de longueur maximale sont appliquées aux éléments DICOM suivants. Pour réussir l'importation, assurez-vous que vos données ne dépassent pas les limites de longueur maximale.

Contraintes relatives aux éléments DICOM lors de l'importation

HealthImaging mot clé	Mot-clé DICOM	clé DICOM	Limite de longueur
DICOM PatientName	Nom du patient	(001 0010)	minimum : 0, maximum : 256
ID de patient DI	Identifiant du patient	(001 0020)	minimum : 0, maximum : 256
DICOM PatientBirthDate	Patient BirthDate	(001 0030)	min : 0, maximum : 18
DICOM PatientSex	PatientSex	(001 0040)	min : 0, maximum : 16
UID DICOM StudyInstance	StudyInstanceUID	(0 020 000D)	min : 0, maximum : 64
DICOM StudyId	ID de l'étude	(002 0010)	min : 0, maximum : 16
DICOM StudyDescription	StudyDescription	(0008 1030)	min : 0, maximum : 64
DICOM NumberOfStudy RelatedSeries	Nombre de séries liées à l'étude	(0020 1206)	minimum : 0, maximum : 10000
DICOM NumberOfStudy RelatedInstances	NumberOfStudyRelatedInstances	(0020 1208)	minimum : 0, maximum : 10000
DICOM Accession Number	AccessionNumber	(0008 0050)	minimum : 0, maximum : 256
DICOM StudyDate	StudyDate	(0008 0020)	min : 0, maximum : 18
DICOM StudyTime	StudyTime	(0008 0030)	min : 0, maximum : 28

Contraintes relatives aux métadonnées DICOM

Lorsque vous mettez `UpdateImageSetMetadata` à jour HealthImaging des attributs de métadonnées, les contraintes DICOM suivantes sont appliquées.

- Impossible de mettre à jour ou de supprimer les attributs privés au niveau du patient/de l'étude/de la série/de l'instance, sauf si la contrainte de mise à jour s'applique aux deux et `updatableAttributes` `removableAttributes`
- Impossible de mettre à jour les attributs HealthImaging générés par AWS suivants : `SchemaVersion` `DatastoreID` `ImageSetIDPixelData`,`Checksum`,`Width`,`Height`,`MinPixelValue`,`MaxPixelValue`, `FrameSizeInBytes`
- Impossible de mettre à jour les attributs DICOM suivants :`Tag.PixelData`,`Tag.StudyInstanceUID`,`Tag.SeriesInstanceUID`, `Tag.SOPInstanceUID` `Tag.StudyID`
- Impossible de mettre à jour les attributs avec le type VR SQ (attributs imbriqués)
- Impossible de mettre à jour les attributs à valeurs multiples
- Impossible de mettre à jour les attributs avec des valeurs incompatibles avec le type VR de l'attribut
- Impossible de mettre à jour les attributs qui ne sont pas considérés comme valides selon la norme DICOM
- Impossible de mettre à jour les attributs entre les modules. Par exemple, si un attribut au niveau du patient est fourni au niveau de l'étude dans la demande de charge utile du client, la demande peut être invalidée.
- Impossible de mettre à jour les attributs si le module d'attribut associé n'est pas présent dans le module existant`ImageSetMetadata`. Par exemple, vous n'êtes pas autorisé à mettre à jour les attributs d'un `seriesInstanceUID` si la série avec n'`seriesInstanceUIDest` pas présente dans les métadonnées des ensembles d'images existants.

Vérification des données des HealthImaging pixels AWS

Lors de l'importation, HealthImaging fournit une vérification intégrée des données de pixels en vérifiant l'état de codage et de décodage sans perte de chaque image. Cette fonctionnalité garantit que les images décodées à l'aide des bibliothèques de décodage HTJ2K correspondent toujours aux images DICOM P10 d'origine importées dans. HealthImaging

- Le processus d'intégration des images commence lorsqu'une tâche d'importation capture l'état de qualité des pixels d'origine des images DICOM P10 avant leur importation. Une somme de contrôle de résolution de trame d'image (IFRC) unique et immuable est générée pour chaque image à l'aide de l'algorithme CRC32. Un IFRC est calculé par niveau de résolution pour les données en pixels de

chaque image. Les valeurs de la somme de contrôle de l'IFRC sont présentées dans le document de métadonnées (`job-output-manifest.json`), triées dans une liste de la base à la résolution complète.

- Une fois les images importées dans un magasin de HealthImaging données et transformées en ensembles d'images, les images codées en HTJ2K sont immédiatement décodées et de nouvelles IFRC sont calculées. HealthImaging compare ensuite les IFRC en pleine résolution des images d'origine aux nouvelles IFRC des images importées pour vérifier l'exactitude.
- Une condition d'erreur descriptive par image correspondante est enregistrée dans le journal des résultats de la tâche d'importation (`job-output-manifest.json`) pour que vous puissiez la consulter et la vérifier.

Pour vérifier les données des pixels

1. Après avoir importé vos données d'imagerie médicale, consultez la description de la réussite (ou de la condition d'erreur) par ensemble d'images enregistrée dans le journal des résultats de la tâche d'importation. `job-output-manifest.json` Pour plus d'informations, consultez Comprendre les tâches d'importation.
2. Les ensembles d'images sont composés de métadonnées et de cadres d'images (données en pixels). Les métadonnées du jeu d'images contiennent des informations sur les cadres d'image associés. Utilisez cette `GetImageSetMetadata` action pour obtenir les métadonnées d'un ensemble d'images. Pour plus d'informations, consultez Obtenir les métadonnées d'un ensemble d'images.
3. `PixelDataChecksumFromBaseToFullResolution`Il contient l'IFRC (somme de contrôle) par niveau de résolution. Vous trouverez ci-dessous un exemple de sortie de métadonnées pour l'IFRC qui est générée dans le cadre du processus d'importation et enregistrée sur`job-output-manifest.json`.

```
"ImageFrames": [{
  "ID": "678906789067890123451234512345123",
  "PixelDataChecksumFromBaseToFullResolution": [
    {
      "Width": 128,
      "Height": 128,
      "Checksum": 2928338830
    },
    {
      "Width": 256,
```

```
      "Height": 256,
      "Checksum": 1362274918
    },
    {
      "Width": 512,
      "Height": 512,
      "Checksum": 2510355201
    }
  ]
```

4. Pour vérifier les données des pixels, accédez à la procédure de vérification des données des pixels GitHub et suivez les instructions du `README.md` fichier pour vérifier de manière indépendante le traitement sans perte des images par bibliothèques de décodage HTJ2K les différents acteurs utilisés par HealthImaging. Au fur et à mesure que les données sont chargées par niveau de résolution, vous pouvez calculer l'IFRC pour les données d'entrée brutes de votre côté et la comparer à la valeur IFRC fournie dans les HealthImaging métadonnées pour cette même résolution afin de vérifier les données en pixels.

bibliothèques de décodage HTJ2K pour AWS HealthImaging

Lors de l'importation, AWS HealthImaging encode toutes les images (données en pixels) au format High-Throughput JPEG 2000 (HTJ2K) sans perte afin de garantir un affichage rapide et constant des images et un accès universel aux fonctionnalités avancées du HTJ2K. Les images étant codées en HTJ2K lors de l'importation, elles doivent être décodées avant d'être visualisées dans une visionneuse d'images.

> **Note**
>
> Le HTJ2K est défini dans la partie 15 de la norme JPEG2000 (ISO/IEC 15444-15:2019). Le HTJ2K conserve les fonctionnalités avancées du JPEG2000, telles que l'évolutivité de la résolution, les limites, le tuilage, la résolution élevée, les canaux multiples et la prise en charge de l'espace colorimétrique.

Rubriques

- bibliothèques de décodage HTJ2K
- Visionneuses d'images

bibliothèques de décodage HTJ2K

En fonction de votre langage de programmation, nous recommandons les bibliothèques de décodage suivantes pour décoder les images.

- NVIDIA NVjpeg2000 : version commerciale, accélérée par GPU
- Logiciel Kakadu — Logiciel commercial, C++ avec liaisons Java et .NET
- OpenJPH — Open source, C++ et WASM
- OpenJPEG — Source libre, C/C++, Java
- openjphpy — Logiciel libre, Python
- pylibjpeg-openjpeg — Source libre, Python

Visionneuses d'images

Vous pouvez afficher les cadres d'image une fois que vous les avez décodés. Les actions HealthImaging d'API AWS prennent en charge diverses visionneuses d'images open source, notamment :

- Open Health Imaging Foundation (OHIF)
- Cornerstone.js

HealthImaging Points de terminaison et quotas AWS

Les rubriques suivantes contiennent des informations sur les points de terminaison et les quotas des HealthImaging services AWS.

Rubriques

- Points de terminaison de service
- Quotas de service

Points de terminaison de service

Un point de terminaison de service est une URL qui identifie un hôte et un port comme point d'entrée d'un service Web. Chaque demande de service web contient un point de terminaison. La plupart AWS des services fournissent des points de terminaison pour des régions spécifiques afin de

permettre une connectivité plus rapide. Le tableau suivant répertorie les points de terminaison de service pour AWS HealthImaging.

Nom de la région	Région	Point de terminaison	Protocole	
US East (Virginie du Nord)	us-east-1	medical-imaging.us-east-1.amazonaws.com	HTTPS	
USA Ouest (Oregon)	us-west-2	medical-imaging.us-west-2.amazonaws.com	HTTPS	
Asie-Pacifique (Sydney)	ap-southeast-2	medical-imaging.ap-southeast-2.amazonaws.com	HTTPS	
Europe (Irlande)	eu-west-1	medical-imaging.eu-west-1.amazonaws.com	HTTPS	

Si vous utilisez des requêtes HTTP pour appeler des HealthImaging actions AWS, vous devez utiliser différents points de terminaison en fonction des actions appelées. Le menu suivant répertorie les points de terminaison de service disponibles pour les requêtes HTTP et les actions qu'ils prennent en charge.

Actions d'API prises en charge pour les requêtes HTTP

data store, import, tagging

Les actions de stockage, d'importation et de balisage de données suivantes sont accessibles via le point de terminaison :

`https://medical-imaging.region.amazonaws.com`

- CreateDatastore
- GetDatastore

- ListDatastores
- DeleteDatastore
- Démarrez DICOM ImportJob
- Obtenez DICOM ImportJob
- Liste DICOM ImportJobs
- TagResource
- ListTagsForResource
- UntagResource

image set

Les actions de définition d'images suivantes sont accessibles via le point de terminaison :

```
https://runtime-medical-imaging.region.amazonaws.com
```

- SearchImageSets
- GetImageSet
- GetImageSetMetadata
- GetImageFrame
- ListImageSetVersions
- UpdateImageSetMetadata

- CopyImageSet
- DeleteImageSet

DICOMweb

HealthImaging propose une représentation du service DICOMWeb Retrieve WADO-RS. Pour plus d'informations, consultez Obtenir une instance DICOM.

Le service DICOMWeb suivant est accessible via un point de terminaison :

```
https://dicom-medical-imaging.region.amazonaws.com
```

- Obtenir une instance DICOM

Quotas de service

Les quotas de service sont définis comme la valeur maximale de vos ressources, actions et éléments de votre AWS compte.

> **Note**
>
> Pour les quotas ajustables, vous pouvez demander une augmentation de quota à l'aide de la console Service Quotas. Pour plus d'informations, consultez Demande d'augmentation de quota dans le Guide de l'utilisateur Service Quotas.

Le tableau suivant répertorie les quotas par défaut pour AWS HealthImaging.

Nom	Par défaut	Ajusta	Description
Nombre maximal de CopyImageSet demandes simultanées par magasin de données	Chaque Région prise en charge : 100	Oui	Nombre maximal de CopyImageSet demandes simultanées par magasin de données dans la AWS région actuelle

Nom	Par défaut	Ajusta	Description
Nombre maximal de DeleteImageSet demandes simultanées par magasin de données	Chaque Région prise en charge : 100	Oui	Nombre maximal de DeleteImageSet demandes simultanées par magasin de données dans la AWS région actuelle
Nombre maximal de UpdateIma geSetMetadata demandes simultanées par magasin de données	Chaque Région prise en charge : 100	Oui	Nombre maximal de UpdateImageSetMeta data demandes simultané es par magasin de données dans la AWS région actuelle
Nombre maximal de tâches d'importa tion simultanées par magasin de données	ap-southeast-2 : 20 Chacune des autres régions prises en charge : 100	Oui	Le nombre maximum de tâches d'importation simultanées par magasin de données dans la AWS région actuelle
Nombre maximum de magasins de données	Par région prise en charge : 10	Oui	Le nombre maximum de magasins de données actifs dans la AWS région actuelle
Nombre maximum de copies ImageFrames autorisées par CopyImageSet demande	Chaque Région prise en charge : 1 000	Oui	Le nombre maximum de copies ImageFram es autorisées par CopyImageSet demande dans la AWS région actuelle

Nom	Par défaut	Ajusta	Description
Nombre maximal de fichiers dans une tâche d'importation DICOM	Chaque région prise en charge : 5 000	Oui	Le nombre maximum de fichiers dans une tâche d'importation DICOM dans la région actuelle AWS
Nombre maximal de dossiers imbriqués dans une tâche d'importation DICOM	Chaque région prise en charge : 10 000	Non	Le nombre maximum de dossiers imbriqués dans une tâche d'importation DICOM dans la région actuelle AWS
Limite de charge utile maximale (en Ko) acceptée par UpdateImageSetMetadata	Chaque région prise en charge : 10 kilo-octets	Oui	La limite de charge utile maximale (en Ko) acceptée UpdateImageSetMetadata dans la région actuelle AWS
Taille maximale (en Go) de tous les fichiers d'une tâche d'importation DICOM	Chaque région prise en charge : 10 gigaoctets	Non	Taille maximale (en Go) de tous les fichiers d'une tâche d'importation DICOM dans la région actuelle AWS
Taille maximale (en Go) de chaque fichier DICOM P10 dans une tâche d'importation DICOM	Chaque région prise en charge : 4 gigaoctets	Non	La taille maximale (en Go) de chaque fichier DICOM P10 dans la tâche d'importation DICOM dans la région actuelle AWS

Nom	Par défaut	Ajusta	Description
Limite de taille maximale (en Mo) ImageSetMetadata par importation, copie et UpdateImageSet	Chaque région prise en charge : 50 mégaoctets	Oui	La limite de taille maximale (en Mo) ImageSetMetadata par importation, copie et UpdateImageSet dans la AWS région actuelle

Limites de HealthImaging limitation d'AWS

Votre AWS compte est soumis à des limites de limitation qui s'appliquent aux actions de HealthImaging l'API AWS. Pour toutes les actions, une `ThrottlingException` erreur est générée si les limites de régulation sont dépassées. Pour plus d'informations, consultez le manuel de référence des HealthImaging API AWS.

> **Note**
>
> Les limites de régulation sont ajustables pour toutes les actions de HealthImaging l'API. Pour demander un ajustement de la limite de régulation, contactez le AWS Support Center.

Le tableau suivant répertorie les limites de limitation pour les actions d' HealthImaging API AWS.

Limites de HealthImaging limitation d'AWS

Action	Fréquence de l'accélérateur	Explosion de l'accélérateur
CreateDatastore	0,085 cuillère à café	1 cuillère à café
GetDatastore	10 TPS	20 c. à thé
ListDatastores	5 c. à thé	10 TPS
DeleteDatastore	0,085 cuillère à café	1 cuillère à café
Démarrez DICOM ImportJob	0,25 cuillère à café	1 cuillère à café
Obtenez DICOM ImportJob	25 c. à thé	50 c. à thé

Action	Fréquence de l'accélérateur	Explosion de l'accélérateur
Liste DICOM ImportJobs	10 TPS	20 c. à thé
SearchImageSets	25 c. à thé	50 c. à thé
GetImageSet	25 c. à thé	50 c. à thé
GetImageSetMetadata	50 c. à thé	100 c. à thé
GetImageFrame	1 000 tps	2000 tps
Obtenir une instance DICOM*	50 c. à thé	100 c. à thé
ListImageSetVersions	25 c. à thé	50 c. à thé
UpdateImageSetMetadata	0,25 cuillère à café	1 cuillère à café
CopyImageSet	0,25 cuillère à café	1 cuillère à café
DeleteImageSet	0,25 cuillère à café	1 cuillère à café
TagResource	10 TPS	20 c. à thé
ListTagsForResource	10 TPS	20 c. à thé
UntagResource	10 TPS	20 c. à thé

*Représentation d'un service DICOMWeb

HealthImaging Exemples de projets AWS

AWS HealthImaging fournit les exemples de projets suivants sur GitHub.

Ingestion du DICOM depuis l'environnement sur site vers AWS HealthImaging

Projet AWS sans serveur visant à déployer une solution IoT Edge qui reçoit des fichiers DICOM d'une source DICOM DIMSE (PACS, VNA, scanner CT) et les stocke dans un compartiment Amazon S3 sécurisé. La solution indexe les fichiers DICOM dans une base de données et met en file d'attente chaque série DICOM à importer dans AWS. HealthImaging Il est composé

d'un composant exécuté à la périphérie qui est géré par AWS IoT Greengrasset d'un pipeline d'ingestion DICOM exécuté dans le AWS cloud.

Proxy TLM (Tile Level Marker)

AWS Cloud Development Kit (AWS CDK)Projet permettant de récupérer des images depuis AWS à l'aide de marqueurs HealthImaging de niveau de tuile (TLM), une fonctionnalité du format JPEG 2000 à haut débit (HTJ2K). Cela se traduit par des temps de récupération plus rapides avec des images à faible résolution. Les flux de travail potentiels incluent la génération de vignettes et le chargement progressif des images.

CloudFront Livraison sur Amazon

Un projet AWS sans serveur visant à créer une CloudFront distribution Amazon avec un point de terminaison HTTPS qui met en cache (à l'aide de GET) et fournit des images depuis le périphérique. Par défaut, le point de terminaison authentifie les demandes à l'aide d'un jeton Web Amazon Cognito JSON (JWT). L'authentification et la signature des demandes sont effectuées à la périphérie à l'aide de Lambda @Edge. Ce service est une fonctionnalité d'Amazon CloudFront qui vous permet d'exécuter du code au plus près des utilisateurs de votre application, ce qui améliore les performances et réduit le temps de latence. Il n'y a aucune infrastructure à gérer.

Interface utilisateur d'AWS HealthImaging Viewer

Un AWS Amplifyprojet de déploiement d'une interface utilisateur frontale avec authentification principale qui vous permet de visualiser les attributs de métadonnées des ensembles d'images et les cadres d'image (données en pixels) stockés dans AWS à HealthImaging l'aide d'un décodage progressif. Vous pouvez éventuellement intégrer le proxy Tile Level Marker (TLM) et/ou les projets Amazon CloudFront Delivery ci-dessus pour charger des cadres d'image à l'aide d'une autre méthode.

Pour consulter d'autres exemples de projets, consultez AWS HealthImaging Samples on GitHub.

Utilisation HealthImaging avec un AWS SDK

AWS des kits de développement logiciel (SDK) sont disponibles pour de nombreux langages de programmation populaires. Chaque SDK fournit une API, des exemples de code et de la documentation qui facilitent la création d’applications par les développeurs dans leur langage préféré.

Documentation SDK	Exemples de code
AWS SDK for C++	AWS SDK for C++ exemples de code
AWS CLI	AWS CLI exemples de code
AWS SDK for Go	AWS SDK for Go exemples de code
AWS SDK for Java	AWS SDK for Java exemples de code
AWS SDK for JavaScript	AWS SDK for JavaScript exemples de code
Kit AWS SDK pour Kotlin	Kit AWS SDK pour Kotlin exemples de code
AWS SDK for .NET	AWS SDK for .NET exemples de code
AWS SDK for PHP	AWS SDK for PHP exemples de code
AWS Tools for PowerShell	Outils pour des exemples PowerShell de code
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) exemples de code
AWS SDK for Ruby	AWS SDK for Ruby exemples de code
Kit AWS SDK pour Rust	Kit AWS SDK pour Rust exemples de code
AWS SDK pour SAP ABAP	AWS SDK pour SAP ABAP exemples de code
Kit AWS SDK pour Swift	Kit AWS SDK pour Swift exemples de code

Exemple de disponibilité

Vous n'avez pas trouvé ce dont vous avez besoin ? Demandez un exemple de code en utilisant le lien Provide feedback (Fournir un commentaire) en bas de cette page.

HealthImaging Versions d'AWS

Le tableau suivant indique à quel moment les fonctionnalités et mises à jour ont été publiées pour le HealthImaging service et la documentation AWS.

Modification	Description	Date

Support amélioré pour les importations de données DICOM non standard	HealthImaging prend en charge les importations de données qui incluent des écarts par rapport à la norme DICOM. Pour plus d'informations, consultez Contraintes relatives aux éléments DICOM.
	• Les éléments de données DICOM suivants peuvent comporter jusqu'à 256 caractères de longueur maximale :
	  • `Patient's Name (0010,0010)`
	  • `Patient ID (0010,0020)`
	  • `Accession Number (0008,0050)`
	• Les variations de syntaxe suivantes sont autorisées pour `Study Instance UIDSeries Instance UID,Treatment Session UID,Manufacturer's Device Class`
		28 juin 2024

	`UID`,`Device UID`, et `Acquisition UID` : • Le premier élément de n'importe quel UID peut être zéro • Les UID peuvent commencer par un ou plusieurs zéros • Les UID peuvent comporter jusqu'à 256 caractères	
Notifications d’événements	HealthImaging s'intègre EventBridge à Amazon pour prendre en charge les applications axées sur les événements. Pour plus d’informations, consultez Utiliser Amazon EventBridge avec HealthImaging.	5 juin 2024
`GetDICOMInstance` pour renvoyer des données DICOM	HealthImaging fournit le `GetDICOMInstance` service permettant de renvoyer les données DICOM Part 10 (`.dcm`fichier) pour un ensemble d'images donné. Pour plus d’informations, consultez Obtenir une instance DICOM.	15 mai 2024

Importation entre comptes	HealthImaging prend en charge les importations de données à partir de compartiments Amazon S3 situés dans d'autres régions prises en charge. Pour plus d'informations, consultez Importation entre comptes pour AWS HealthImaging.	15 mai 2024
Améliorations de la recherche pour les ensembles d'images	HealthImaging `SearchImageSets` action prend en charge les améliorations de recherche suivantes. Pour plus d'informations, consultez Recherche de séries d'images. • Support supplémentaire pour la recherche sur `UpdatedAt` et `SeriesInstanceUID` • Recherche entre l'heure de début et l'heure de fin • Trier les résultats de recherche par `Ascending` ou `Descending` • Les paramètres de la série DICOM sont renvoyés dans les réponses	3 avril 2024

Augmentation de la taille maximale des fichiers pour les importations	HealthImaging prend en charge une taille de fichier maximale de 4 Go pour chaque fichier DICOM P10 dans une tâche d'importation. Pour plus d'informations, consultez Quotas de service.	6 mars 2024
Syntaxes de transfert pour JPEG Lossless et HTJ2K	HealthImaging prend en charge les syntaxes de transfert suivantes pour les importations de tâches. Pour plus d'informations, consultez Syntaxes de transfert prises en charge. • 1.2.840.10008.1.2.4.57 — JPEG sans perte non hiérarchique (processus 14) • 1.2.840.10008.1.2.4.201 — Compression d'image JPEG 2000 à haut débit (sans perte uniquement) • 1.2.840.10008.1.2.4.202 — JPEG 2000 à haut débit avec options RPCL (compression d'image sans perte uniquement) • 1.2.840.10008.1.2.4.203 — Compression d'image JPEG 2000 à haut débit	16 février 2024

Exemples de code testés	HealthImaging la documentation fournit des exemples de code testés AWS CLI et AWS des SDK pour Python JavaScript, Java et C++. Pour plus d'informations, consultez Exemples de code pour HealthImaging l'utilisation des AWS SDK.	19 décembre 2023
Augmentation du nombre maximum de fichiers pour les importations	HealthImaging prend en charge jusqu'à 5 000 fichiers pour une seule tâche d'importation. Pour plus d'informations, consultez Quotas de service.	19 décembre 2023
Dossiers imbriqués pour les importations	HealthImaging prend en charge jusqu'à 10 000 dossiers imbriqués pour une seule tâche d'importation. Pour plus d'informations, consultez Quotas de service.	1er décembre 2023
Importations plus rapides	HealthImaging fournit des importations 20 fois plus rapides dans toutes les régions prises en charge. Pour plus d'informations, consultez Points de terminaison de service.	1er décembre 2023

CloudFormation soutien	HealthImaging prend en charge l'infrastructure sous forme de code (IaC) pour le provisionnement des magasins de données. Pour plus d'informations, consultez Création de HealthImaging ressources AWS avec AWS CloudFormation.	21 septembre 2023
Disponibilité générale	AWS HealthImaging est disponible pour tous les clients des régions des États-Unis est (Virginie du Nord), de l'ouest des États-Unis (Oregon), de l'Europe (Irlande) et de l'Asie-Pacifique (Sydney).	26 juillet 2023

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.