Amazon HealthLake

Amazon HealthLake Developer Guide

Amazon HealthLake: Amazon HealthLake Developer Guide

Table of Contents

What is Amazon HealthLake?

Amazon HealthLake is a HIPAA-eligible service that healthcare providers, health insurance companies, and pharmaceutical companies can use to store, transform, query, and analyze large-scale health data.

Health data is frequently incomplete and inconsistent. It's also often unstructured, with information contained in clinical notes, lab reports, insurance claims, medical images, recorded conversations, and time-series data (for example, heart ECG or brain EEG traces).

Healthcare providers can use HealthLake to store, transform, query, and analyze data in the AWS Cloud. Using the HealthLake integrated medical natural language processing (NLP) capabilities, you can analyze unstructured clinical text from diverse sources. HealthLake transforms unstructured data using natural language processing models, and provides powerful query and search capabilities. You can use HealthLake to organize, index, and structure patient information in a secure, compliant, and auditable manner.

Benefits of Amazon HealthLake

With Amazon HealthLake, you can:

- **Quickly and easily ingest health data** – You can bulk import on-premises Fast Healthcare Interoperability Resources (FHIR) files, including clinical notes, lab reports, insurance claims, and more, to an Amazon Simple Storage Service (Amazon S3) bucket. You can then use the data in downstream applications or workflows.
- **Store your data in the AWS Cloud in a secure, HIPAA-eligibile manner that can be audited**– You can store data in FHIR format, so it can be easily queried. HealthLake creates a complete, chronological view of each patient's medical history, and structures it in the R4 FHIR standard format.
- **Transform unstructured data using specialized ML models** – Integrated medical natural language processing (NLP) transforms all of the raw medical text data using specialized ML models that have been trained to understand and extract meaningful information from unstructured healthcare data. With integrated medical NLP, you can automatically extract entities (for example, medical procedures and medications), entity relationships (for example, a medication and its dosage), and entity traits (for example, positive or negative test result or time of procedure) data from your medical text.
- **Use powerful query and search capabilities** – HealthLake supports FHIR CRUD (Create/Read/Update/Delete) and FHIR Search operations.

Amazon HealthLake use cases

You can use Amazon HealthLake for the following healthcare applications:

- **Population health management** – HealthLake helps healthcare organizations analyze population health trends, outcomes, and costs. This helps organization to identify the most appropriate intervention for a patient population, and choose better care management options.
- **Improving quality of care** – HealthLake aids hospitals, health insurance companies, and life sciences organizations close gaps in care, improve quality of care, and reduce cost by compiling a complete view of a patient's medical history.
- **Optimize hospital efficiency** – HealthLake offers hospitals key analytics and machine learning tools to improve efficiency and reduce hospital waste.

Accessing Amazon HealthLake

You can access Amazon HealthLake through the AWS Management Console, AWS Command Line Interface (AWS CLI), or the AWS SDKs.

1. AWS Management Console – Provides a web interface that you can use to access HealthLake.
2. AWS Command Line Interface (AWS CLI) – Provides commands for a broad set of AWS services, including HealthLake, and is supported on Windows, macOS, and Linux. For more information about installing the AWS CLI, see AWS Command Line Interface
3. AWS SDKs – AWS provides SDKs (software development kits) that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, and so on). The SDKs provide a convenient way to create programmatic access to HealthLake and AWS. For more information, see the AWS SDK for Python

HIPAA eligibility and data security

This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see HIPAA Overview.

Connections to HealthLake containing PHI or personally identifiable information (PII) must be encrypted. By default, all connections to HealthLake use HTTPS over TLS. HealthLake stores encrypted customer content and operates by the AWS Shared Responsibility principle.

Pricing

For information about HealthLake pricing, see the Amazon HealthLake pricing page.

How Amazon HealthLake works

Amazon HealthLake maintains Data Stores of health records in FHIR-compliant format. You can perform the following tasks using the Amazon HealthLake console, AWS Command Line Interface (AWS CLI), or APIs:

- Create, monitor, and delete a Data Store
- Import your data from an Amazon Simple Storage Service (Amazon S3) bucket into the Data Store
- Query data using `Create`, `Read`, `Update`, and `Delete` functions
- Use FHIR search functionality
- Transform your data using integrated medical natural language processing (NLP)

Creating and monitoring Data Stores

With Amazon HealthLake, you can create and manage Data Stores for storing R4 FHIR Resources. Use create-fhir-datastore to create a new Data Store, describe-fhir-datastore to learn more about the properties of a Data Store, and list-fhir-datastore to see all Data Stores associated with your account and their status. When a Data Store is no longer needed, you can delete it using delete-fhir-datastore.

Create, Read, Update, Delete (CRUD) operations

Manage and query data using the `CreateResource`, `ReadResource`, `UpdateResource`, and `DeleteResource` operations for 71 different FHIR resource types. For a list of FHIR resource types supported by , see Managing FHIR resources (p. 48). These operations are all handled through an HTTP client.

Integrated medical natural language processing (NLP)

HealthLake has integrated medical natural language processing for the `DocumentReference` resource type. HealthLake automatically runs the integrated medical NLP on all `DocumentReference` resources as they are written to the system in an `Import` operation, during a `Create` operation, or during an `Update` operation. The original resource stays unchanged, and the extracted medical information is automatically appended as custom fields. These fields are indexed and queryable after extraction.

FHIR search functionality

You can search health records that are stored in the Data Store either through a specific resource type with supported search parameters or for resource IDs in the server without specifying the resource type. When a FHIR record is created, it is first saved into a FHIR Data Store, where it can be read, updated, queried, or deleted. After the data is ingested, it is indexed using OpenSearch Service, which makes the data searchable. HealthLake operates with eventual consistency with the Data Store, meaning that there might be a brief latency before the data is searchable within the Data Store.

Import data

Amazon HealthLake enables you to bulk import your files from an Amazon S3 bucket. Use either the console or start-fhir-import-job to begin an import job. Afterwards, you can use describe-fhir-import-job to monitor the status of the job and discover its properties. After the import job is complete, the data can then be added to a Data Store, transformed, or analyzed and used in downstream applications.

Export data

Amazon HealthLake enables you to bulk export your files to an Amazon S3 bucket. Use either the console or start-fhir-export-job to begin an export job. Afterwards, you can use describe-fhir-export-job to monitor the status of the job and discover its properties. After the export job is complete, the data can then be visualized using AWS Quicksight or accessed by other AWS services.

Getting started with Amazon HealthLake

To get started using Amazon HealthLake, set up an AWS account and create an AWS Identity and Access Management (IAM) user. To use the AWS Command Line Interface, AWS SDK for Python, or the AWS SDK for Java, download and configure them.

Account set up with Amazon HealthLake

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services. For access to Amazon HealthLake during the public preview, you need to request access through the AWS Management Console first.

If you are a new AWS customer, you can get started with Amazon HealthLake for free. For more information, see AWS Free Usage Tier.

If you already have an AWS account, skip to the next section.

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Record your AWS account ID because you'll need it for the next task.

Create an IAM User

Services in AWS, such as Amazon HealthLake, require that you provide credentials to access them. This allows the service to determine whether you have permissions to access the service's resources.

We strongly recommend that you access AWS using AWS Identity and Access Management (IAM), not the credentials for your AWS account. To use IAM to access AWS, create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user. You can then access AWS using a special URL and the IAM user's credentials.

The getting started exercises in this guide assume that you have a user with administrator privileges, `adminuser`.

To create an administrator and sign in to the console

1. Create a user named `adminuser` in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. Sign in to the AWS Management Console using a special URL. For more information, see How Users Sign In to Your Account in the *IAM User Guide*.
3. To ensure that all necessary users and roles have access to HealthLake, attach a permission policy to grant access to the service. The following is an example granting access to HealthLake.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "healthlake:*"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

For more information about IAM, see the following:

- AWS Identity and Access Management (IAM)
- Getting started
- IAM User Guide

Next step: Setting up the AWS CLI

Set up the AWS Command Line Interface (AWS CLI)

You don't need the AWS CLI to perform the steps in all the getting started exercises. However, some of the other exercises in this guide do require it.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:
   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface
2. In the AWS CLI `config` file, add a named profile for the administrator.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   You use this profile when running the AWS CLI commands. Under the security principle of least privilege, we recommend that you create a separate IAM role with privileges specific to the tasks being performed. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*. For a list of AWS Regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.
3. Verify the setup by typing the following help command at the command prompt.

   ```
   aws healthlake help
   ```

   If the AWS CLI is configured correctly, you will see a brief description of Amazon HealthLake and a list of available commands.

4. Amazon HealthLake is available only in the US East Northern Virginia (us-east-1) Region. You don't need to specify the "--endpoint" option when using the AWS CLI, but you can if necessary. The endpoint is https://aws68918.us-east-1.amazonaws.com/.

   For example, to list all of the HealthLake FHIR Data Stores that you own, you use the following command.

```
$ aws healthlake list-fhir-datastores \
```

Creating a FHIR Data Store using the AWS Command Line Interface

The following example demonstrates using the `CreateFHIRDatastore` operation with the AWS CLI. To run the example, you must install the AWS CLI.

The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

```
aws healthlake create-fhir-datastore \
    --datastore-type-version R4 \
    --preload-data-config PreloadDataType="SYNTHEA" \
    --datastore-name "FhirTestDatastore"
```

Creating a FHIR Data Store using the SDK for Python

The following example demonstrates using the `CreateFHIRDatastore` operation using the AWS SDK for Python.

```
import boto3
healthlake = boto3.client(service_name='healthlake',
                          region_name='us-east-1',
                          use_ssl=True)
DataStore_type_version = "R4"
DataStore_name = "TestDatastore123"
preload_type = "SYNTHEA"
preload_option = {'PreloadDataType': preload_type}
print('Calling CreateFHIRDatastore\n')
create_response = healthlake.create_fhir_datastore(
    DatastoreTypeVersion=DataStore_type_version,
    DatastoreName=DataStore_name,
    PreloadDataConfig=preload_option)
print("Create FHIR Datastore response: \n", create_response)
print('End of CreateFHIRDatastore\n')
```

Creating a FHIR Data Store using the AWS SDK for Java

The following example uses the `CreateFHIRDatastore` operation with Java. To run the example, install the AWS SDK for Java. For instructions on installing the AWS SDK for Java, see Set up the AWS SDK for Java.

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;

import com.amazonaws.services.HealthLake.AWSHealthLake;
import com.amazonaws.services.HealthLake.AWSHealthLakeClient;
import com.amazonaws.services.HealthLake.model.CreateFHIRDatastoreRequest;
import com.amazonaws.services.HealthLake.model.CreateFHIRDatastoreResult;
import com.amazonaws.services.HealthLake.model.DescribeFHIRDatastoreRequest;
import com.amazonaws.services.HealthLake.model.DescribeFHIRDatastoreResult;
import com.amazonaws.services.HealthLake.model.FHIRVersion;
import com.amazonaws.services.HealthLake.model.ListFHIRDatastoresRequest;
import com.amazonaws.services.HealthLake.model.ListFHIRDatastoresResult;
import com.amazonaws.services.HealthLake.model.PreloadDataConfig;
import com.amazonaws.services.HealthLake.model.PreloadDataType;

public class App{

    public static void main( String[] args ) {

        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
        AWSCredentialsProvider awsCreds = DefaultAWSCredentialsProviderChain.getInstance();

        AWSHealthLake awsHealthLake = AWSHealthLakeClient.builder()
            .withRegion("us-east-1").withCredentials(awsCreds).defaultClient();

        CreateFHIRDatastoreRequest createFHIRDatastoreRequest = new
 CreateFHIRDatastoreRequest()
            .withData StoreName("TestDatastore123")
            .withData StoreTypeVersion(FHIRVersion.R4)
            .withPreloadDataConfig(new PreloadDataConfig()
                .withPreloadDataType(PreloadDataType.SYNTHEA));



    }
}
```

Getting started with an HTTP client

The Amazon HealthLake FHIR APIs used to create, read, update, and delete FHIR resources aren't supported by the AWS SDK for Python (Boto). You must use these Representational State Transfer (REST) APIs through a REST HTTP client.

Preloaded datatypes

HealthLake supports only SYNTHEA as a preloaded data type. Synthea is a synthetic patient generator that models the medical history of model-generated patients. It's an open-source Git repository that allows HealthLake to generate FHIR R4-compliant resource bundles so that users can test models without using actual patient data.

The following resource types are available as preloaded Data Stores.

Supported Synthea resource types

AllergyIntolerance	Location
CarePlan	MedicationAdministration
CareTeam	MedicationRequest

Claim	Observation
Condition	Organization
Device	Patient
DiagnosticReport	Practitioner
Encounter	PractitionerRole
ExplanationofBenefit	Procedure
ImagingStudy	Provenance
Immunization	

Security in Amazon HealthLake

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to HealthLake, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using HealthLake. The following topics show you how to configure HealthLake to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your HealthLake resources.

Topics

Data Protection in Amazon HealthLake

The AWS shared responsibility model applies to data protection in Amazon HealthLake. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with HealthLake or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest for Amazon HealthLake

HealthLake provides encryption by default to protect sensitive customer data at rest by using a service owned AWS Key Management Service (AWS KMS) key. Customer-managed KMS keys are also supported and are required for both importing and exporting files from a Data Store. To learn more about Customer-managed KMS Key, see Amazon Key Management Service. Customers can choose an AWS owned KMS key or a Customer-managed KMS key when creating a Data Store. The encryption configuration cannot be changed after a Data Store has been created. If a Data Store is using an AWS owned KMS Key, it will be denoted as AWS_OWNED_KMS_KEY and you will not see the specific key used for encryption at rest.

AWS owned KMS key

HealthLake uses these keys by default to automatically encrypt potentially sensitive information such as personally identifiable or Private Health Information(PHI) data at rest. AWS owned KMS keys aren't stored in your account. They're part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned KMS keys to protect your data. You can't view, manage, use AWS owned KMS keys, or audit their use. However, you don't need to do any work or change any programs to protect the keys that encrypt your data.

You're not charged a monthly fee or a usage fee if you use AWS owned KMS keys, and they don't count against AWS KMS quotas for your account. For more information, see AWS owned keys.

Customer managed KMS keys

HealthLake supports the use of a symmetric customer managed KMS key that you create, own, and manage to add a second layer of encryption over the existing AWS owned encryption. Because you have full control of this layer of encryption, you can perform such tasks as:

- Establishing and maintaining key policies, IAM policies, and grants
- Rotating key cryptographic material
- Enabling and disabling key policies
- Adding tags
- Creating key aliases

- Scheduling keys for deletion

You can also use CloudTrail to track the requests that HealthLake sends to AWS KMS on your behalf. Additional AWS KMS charges apply.For more information, see customer owned keys.

Create a customer managed key

You can create a symmetric customer managed key by using the AWS Management Console, or the AWS KMS APIs.

Follow the steps for Creating symmetric customer managed key in the AWS Key Management Service Developer Guide.

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it. When you create your customer managed key, you can specify a key policy. For more information, see Managing access to customer managed keys in the AWS Key Management Service Developer Guide.

To use your customer managed key with your HealthLake resources, kms:CreateGrant operations must be permitted in the key policy. This adds a grant to a customer managed key which controls access to a specified KMS key, which gives a user access to the kms:grant operations HealthLake requires. See Using grantsfor more information.

To use your customer managed KMS key with your HealthLake resources, the following API operations must be permitted in the key policy:

- kms:CreateGrant adds grants to a specific customer managed KMS key which allows access to grant operations.
- kms:DescribeKey provides the customer managed key details needed to validate the key. This is required for all operations.
- kms:GenerateDataKey provides access to encrypt resources at rest for all write operations.
- kms:Decrypt provides access to read or search operations for encrypted resources.

The following is a policy statement example that allows a user to create and interact with a Data Store in Amazon HealthLake which is encrypted by that key:

```
"Statement": [
    {
        "Sid": "Allow access to create data stores and do CRUD/search in Amazon
 HealthLake",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::111122223333:HealthLakeFullAccessRole"
        },
        "Action": [
            "kms:DescribeKey",
            "kms:CreateGrant",
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:ViaService": "healthlake.amazonaws.com",
                "kms:CallerAccount": "111122223333"
            }
```

```
        }
    }
]
```

Required IAM permissions for using a customer managed KMS key

When creating a Data Store with AWS KMS encryption enabled using a customer managed KMS key, there are required permissions for both the key policy and the IAM policy for the user or role creating the HealthLake Data Store.

You can use the kms:ViaService condition key to limit use of the KMS key to only requests that originate from HealthLake.

For more information about key policies, see Enabling IAM policies in the AWS Key Management Service Developer Guide.

The IAM user, IAM role, or AWS account creating your repositories must have the kms:CreateGrant,kms:GenerateDataKey, and kms:DescribeKey permissions plus the necessary HealthLake permissions.

How HealthLake uses grants in AWS KMS

HealthLake requires a grant to use your customer managed KMS key. When you create a Data Store encrypted with a customer managed KMS key, HealthLake creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give HealthLake access to a KMS key in a customer account.

The grants that HealthLake creates on your behalf should not be revoked or retired. If you revoke or retire the grant that gives HealthLake permission to use the AWS KMS keys in your account, HealthLake cannot access this data, encrypt new FHIR resources pushed to the Data Store, or decrypt them when they are pulled. When you revoke or retire a grant for HealthLake, the change occurs immediately. To revoke access rights, you should delete the Data Store rather than revoking the grant. When a Data Store is deleted, HealthLake retires the grants on your behalf.

Monitoring your encryption keys for HealthLake

You can use CloudTrail to track the requests that HealthLake sends to AWS KMS on your behalf when using a customer managed KMS key. The log entries in the CloudTrail log show healthlake.amazonaws.com in the userAgent field to clearly distinguish requests made by HealthLake.

The following examples are CloudTrail events for CreateGrant, GenerateDataKey, Decrypt, and DescribeKey to monitor AWS KMS operations called by HealthLake to access data encrypted by your customer managed key.

The following shows how to use CreateGrant to allow HealthLake to access a customer provided KMS key, enabling HealthLake to use that KMS key to encrypt all customer data at rest.

Users are not required to create their own grants. HealthLake creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give HealthLake access to a AWS KMS key in a customer account.

```
        {
    "eventVersion": "1.08",
    "userIdentity": {
```

```
        "type": "AssumedRole",
        "principalId": "EXAMPLEROLE:Sampleuser01",
        "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": "2021-06-30T19:33:37Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "healthlake.amazonaws.com"
    },
    "eventTime": "2021-06-30T20:31:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "healthlake.amazonaws.com",
    "userAgent": "healthlake.amazonaws.com",
    "requestParameters": {
        "operations": [
            "CreateGrant",
            "Decrypt",
            "DescribeKey",
            "Encrypt",
            "GenerateDataKey",
            "GenerateDataKeyWithoutPlaintext",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
        ],
        "granteePrincipal": "healthlake.us-east-1.amazonaws.com",
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE_KEY_ARN",
        "retiringPrincipal": "healthlake.us-east-1.amazonaws.com"
    },
    "responseElements": {
        "grantId": "EXAMPLE_ID_01"
    },
    "requestID": "EXAMPLE_ID_02",
    "eventID": "EXAMPLE_ID_03",
    "readOnly": false,
    "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"
}
```


The following examples shows how to use GenerateDataKey to ensure the user has necessary permissions to encrypt data before storing it.

```
          {
    "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": "healthlake.amazonaws.com"
    },
    "eventTime": "2021-06-30T21:17:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "healthlake.amazonaws.com",
    "userAgent": "healthlake.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"
}
```


The following example shows how HealthLake calls the Decrypt operation to use the stored encrypted data key to access the encrypted data.


```
          {
    "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": "healthlake.amazonaws.com"
    },
    "eventTime": "2021-06-30T21:21:59Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "healthlake.amazonaws.com",
    "userAgent": "healthlake.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"
}
```


The following example shows how HealthLake uses the DescribeKey operation to verify if the AWS KMS customer owned AWS KMS key is in a usable state and to help a user troubleshoot if it is not functional.


```
          {
    "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": "healthlake.amazonaws.com"
    },
    "eventTime": "2021-07-01T18:36:36Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "healthlake.amazonaws.com",
    "userAgent": "healthlake.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"
}
```


Learn more

The following resources provide more information about data at rest encryption.

For more information about AWS Key Management Service basic concepts, see the AWS KMS documentation.

For more information about Security best practices in the AWS KMS documentation.

Encryption in transit for Amazon HealthLake

Amazon HealthLake uses TLS 1.2 to encrypt data in transit through the public endpoint and through backend services.

Identity and Access Management for Amazon HealthLake

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and

authorized (have permissions) to use HealthLake resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in HealthLake.

Service user – If you use the HealthLake service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more HealthLake features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in HealthLake, see Troubleshooting Amazon HealthLake identity and access (p. 28).

Service administrator – If you're in charge of HealthLake resources at your company, you probably have full access to HealthLake. It's your job to determine which HealthLake features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with HealthLake, see How Amazon HealthLake works with IAM (p. 22).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to HealthLake. To view example HealthLake identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon HealthLake (p. 26).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for Amazon HealthLake in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that

you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon HealthLake works with IAM

Before you use IAM to manage access to HealthLake, learn what IAM features are available to use with HealthLake.

IAM features you can use with Amazon HealthLake

IAM feature	HealthLake support
Identity-based policies (p. 22)	Yes
Resource-based policies (p. 23)	Yes
Policy actions (p. 23)	Yes
Policy resources (p. 24)	Yes
Policy condition keys (p. 24)	Yes
ACLs (p. 25)	No
ABAC (tags in policies) (p. 25)	Yes
Temporary credentials (p. 25)	Yes
Principal permissions (p. 25)	Yes
Service roles (p. 26)	Yes
Service-linked roles (p. 26)	No

To get a high-level view of how HealthLake and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon HealthLake

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon HealthLake

To view examples of HealthLake identity-based policies, see Identity-based policy examples for Amazon HealthLake (p. 26).

Resource-based policies within Amazon HealthLake

Supports resource-based policies	Yes

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Amazon HealthLake

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of HealthLake actions, see Actions defined by Amazon HealthLake in the *Service Authorization Reference*.

Policy actions in HealthLake use the following prefix before the action:

```
healthlake
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "healthlake:action1",
      "healthlake:action2"
         ]
```

To view examples of HealthLake identity-based policies, see Identity-based policy examples for Amazon HealthLake (p. 26).

Policy resources for Amazon HealthLake

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of HealthLake resource types and their ARNs, see Resources defined by Amazon HealthLake in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon HealthLake.

To view examples of HealthLake identity-based policies, see Identity-based policy examples for Amazon HealthLake (p. 26).

Policy condition keys for Amazon HealthLake

Supports policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of HealthLake condition keys, see Condition keys for Amazon HealthLake in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon HealthLake.

To view examples of HealthLake identity-based policies, see Identity-based policy examples for Amazon HealthLake (p. 26).

Access control lists (ACLs) in Amazon HealthLake

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Amazon HealthLake

Supports ABAC (tags in policies)	Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Amazon HealthLake

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Amazon HealthLake

Supports principal permissions	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for Amazon HealthLake in the *Service Authorization Reference*.

Service roles for Amazon HealthLake

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

Warning
Changing the permissions for a service role might break HealthLake functionality. Edit service roles only when HealthLake provides guidance to do so.

Service-linked roles for Amazon HealthLake

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon HealthLake

By default, IAM users and roles don't have permission to create or modify HealthLake resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete HealthLake resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using HealthLake quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Amazon HealthLake console

To access the Amazon HealthLake console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the HealthLake resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

To ensure that users and roles can still use the HealthLake console, also attach an IAM policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*. Users can manage access to the console through the service managed policies **AmazonHealthLakeFullAccess** and **AmazonHealthLakeReadOnlyAccess**.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "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": "*"
        }
    ]
}
```

Troubleshooting Amazon HealthLake identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with HealthLake and IAM.

Topics

I am not authorized to perform an action in Amazon HealthLake

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `healthlake:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 healthlake:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-widget`* resource using the `healthlake:`*`GetWidget`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to HealthLake.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in HealthLake. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access Amazon HealthLake

To allow others to access HealthLake, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in HealthLake.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Amazon HealthLake resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether HealthLake supports these features, see How Amazon HealthLake works with IAM (p. 22).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.

- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging Amazon HealthLake API Calls with AWS CloudTrail

Amazon HealthLake is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in HealthLake. CloudTrail captures all API calls for HealthLake as events. The calls captured include calls from the HealthLake console and code calls to the HealthLake API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for HealthLake. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to HealthLake, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon HealthLake Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in HealthLake, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for HealthLake, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All HealthLake actions are logged by CloudTrail and are documented in the HealthLake API reference and in this Developer Guide for actions performed through an HTTP client. For example, calls to the following actions generate entries in the CloudTrail log files:

- `DescribeFHIRImportJob`
- `DescribeFHIRExportJob`
- `StartFHIRImportJob`
- `ListFHIRImportJobs`
- `StartFHIRExportJob`

- `ListFHIRExportJobs`
- `CreateFHIRDatastore`
- `ListFHIRDatastores`
- `DeleteFHIRDatastore`
- `DescribeFHIRDatastore`
- `UpdateResource`
- `CreateResource`
- `DeleteResource`
- `ReadResource`
- `GetCapabilities`
- `SearchWithGet`
- `SearchWithPost`

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding Amazon HealthLake Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateFHIRDatastore` action.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA2B3ZHOADD2OJ4AHJX:git full_access_iam_role580074395690222150",
        "arn": "arn:aws:sts::691207106566:assumed-role/
colossusfrontend_full_access_iam_role/_iam_role580074395690222150",
        "accountId": "AccountID",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA2B3ZHOADD2OJ4AHJX",
                "arn": "arn:aws:iam::691207106566:role/full_access_iam_role",
                "accountId": "AccountID",
                "userName": "full_access_iam_role"
            },
            "webIdFederationData": {

            },
```

```
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-11-20T00:08:15Z"
            }
        }
    },
    "eventTime": "2020-11-20T00:08:16Z",
    "eventSource": "healthlake.amazonaws.com",
    "eventName": "CreateFHIRDatastore",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "3.213.247.1",
    "userAgent": "Coral/Netty4",
    "requestParameters": {
        "datastoreName":
 "testCreateFHIRDatastore_GBYAZFCLLBLSUTOYYFQZRLBLQJNFOYQVRPZBOJAIIUAHICAEAGIWLNVQEYAMSXVWMBLXCDCLMJKYF
        "datastoreTypeVersion": "R4",
        "clientToken": "d737ffe0-14dd-44cc-9f0a-fdf59b26c66b"
    },
    "responseElements": {
        "datastoreId": "datastoreID",
        "datastoreArn": "arn:aws:healthlake:us-
east-1:691207106566:datastore/55576c487ff4975262b10d1d65eb4509",
        "datastoreStatus": "CREATING",
        "datastoreEndpoint": "datastore_endpoint/"
    },
    "requestID": "68e62bdd-d2d4-44c1-af69-e6f055a69f99",
    "eventID": "7ef483dc-5dca-469e-823a-7d9e3a7fe924",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "691207106566"
}
```


Compliance Validation for Amazon HealthLake

Third-party auditors assess the security and compliance of Amazon HealthLake as part of multiple AWS compliance programs. For HealthLake this includes HIPAA.

To learn whether HealthLake or other AWS services are in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

  Note
  Not all services are compliant with HIPAA.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.

- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon HealthLake

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, HealthLake offers several features to help support your data resiliency and backup needs.

Infrastructure Security in Amazon HealthLake

As a managed service, Amazon HealthLake is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access HealthLake through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Security best practices in Amazon HealthLake

Amazon HealthLake provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

- Implement least privilege access.
- Whenever possible, use Customer-Managed-Keys(CMKs) to encrypt your data. To learn more about CMKs, see Amazon Key Management Service.
- Use Search with POST, not Search with GET when querying for PHI or PII in your Data Store.
- Limit access to sensitive and important auditing functions.
- When creating resources through the update or bulk import APIs, do not use PHI or PII, including the names of Data Stores and jobs, in any visible fields or in the logical FHIR ID (LID).

- When sending create, read, update, delete, or search requests, do not use PHI in the HTTP header.
- Enable AWS CloudTrail to audit Amazon HealthLake use and to ensure that there is no unexpected activity.
- Review best practices for using Amazon S3 buckets securely. To learn more, see Security best practices in the *Amazon S3 user guide*.

Amazon HealthLake and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon HealthLake by creating an *interface VPC endpoint*. Interface VPC endpoints are powered by AWS PrivateLink, a technology that you can use to privately access HealthLake APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with HealthLake APIs. Traffic between your VPC and HealthLake does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for HealthLake VPC endpoints

Before you set up an interface VPC endpoint for HealthLake, be sure you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

HealthLake supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for HealthLake

You can create a VPC endpoint for the HealthLake service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for HealthLake using the following service name:

- com.amazonaws.*region*.healthlake

If you turn on private DNS for the endpoint, you can make API requests to HealthLake using its default DNS name for the Region. For example, `healthlake.us-east-1.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for HealthLake

You can attach an endpoint policy to your VPC endpoint that controls access to HealthLake. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for HealthLake actions

The following is an example of an endpoint policy for HealthLake. When attached to an endpoint, this policy grants access to the HealthLake `CreateFHIRDatastore` action for all principals on all resources.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "healthlake:create-fhir-datastore"
         ],
         "Resource":"*"
      }
   ]
}
```

Tagging resources in Amazon HealthLake

Important notice

Amazon HealthLake protects customer data under the AWS Shared Responsibility Model policies. This means that all customer data is encrypted both in transition and at-rest. However, not all customer-inputed names for Data Stores or job-based operations are encypted. They should never contain Personally Identifiable Information or Protected Health Information. For more information, see the Amazon HealthLake Security chapter.

Tagging using HealthLake resources

You can assign metadata to your AWS resources in the form of *tags*. Each tag is a label consisting of a user-defined key and value. Tags can help you manage, identify, organize, search for, and filter resources.

This topic describes commonly used tagging categories and strategies to help you implement a consistent and effective tagging strategy. The following sections assume basic knowledge of AWS resources, tagging, detailed billing, and AWS Identity and Access Management (IAM).

Each tag has two parts:

- A *tag key* (for example, CostCenter, Environment, or Project). Tag keys are case sensitive.
- A *tag value* (for example, 111122223333 or Production). Like tag keys, tag values are case sensitive.

You can use tags to categorize resources by purpose, owner, environment, or other criteria. For more information, see AWS Tagging Strategies.

You can add, change, or remove tags one resource at a time from each resource's service console, service API, or the AWS CLI.

To enable tagging, make sure TagResources are authorized. You can authorize TagResources by attaching an IAM policy like the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "healthlake:CreateFHIRDatastore",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "healthlake:TagResource",
            "Resource": "*"
        }
    ]
```

```
}
```

Best practices

As you create a tagging strategy for AWS resources, follow best practices:

- Do not store Personally Identifiable Information (PII), Personal Health Information(PHI) or other sensitive information in tags.
- Use a standardized, case-sensitive format for tags, and apply it consistently across all resource types.
- Consider tag guidelines that support multiple purposes, like managing resource access control, cost tracking, automation, and organization.
- Use automated tools to help manage resource tags. AWS Resource Groups and the Resource Groups Tagging API enable programmatic control of tags, making it possible to automatically manage, search, and filter tags and resources.
- Tagging is more effective when you use more tags.
- Tags can be edited or modified as user needs change, however to update access control tags, you must also update the policies that reference those tags to control access to your resources.

Tagging requirements

Tags have the following requirements:

- Keys can't be prefixed with aws:.
- Keys must be unique per tag set.
- A key must be between 1 and 128 allowed characters.
- A value must be between 0 and 256 allowed characters.
- Values do not need to be unique per tag set.
- Allowed characters for keys and values are Unicode letters, digits, white space, and any of the following symbols: _ . : / = + - @.
- Keys and values are case sensitive.

Adding a tag to a Data Store

Adding tags to a Data Store can help you identify and organize your AWS resources and manage access to them. First, you add one or more tags (key-value pairs) to a Data Store. You can use up to fifty tags per user. There are also restrictions on the characters you can use in the key and value fields.

After you have tags, you can create IAM policies to manage access to the Data Store based on these tags. You can use the the HealthLake console or the AWS CLI to add tags to a Data Store. Adding tags to a repository can impact access to that repository. Before you add a tag to a Data Store, make sure to review any IAM policies that might use tags to control access to resources such as Data Stores.

Follow these steps to use the AWS CLI to add a tag to a HealthLake Data Store. To add a tag to a Data Store when you create it, see Creating and monitoring FHIR resources (p. 45).

At the terminal or command line, run the tag-resource command, specifying the Amazon Resource Name (ARN) of the Data Store where you want to add tags and the key and value of the tag you want to add. You can add more than one tag to a Data Store. There are also restrictions on the characters you can use in the key and value fields, as listed in Tagging requirements (p. 38)For example, to add tags to a Data

Store while it is being created, you would use the following command in the AWS CLI. The name of the Data Store is Test_Data_Store, and the two added tags with keys are key1 and key2 with values as value1 and value2 respectively
:

```
aws healthlake create-fhir-datastore --datastore-type-version R4 --preload-data-config
 PreloadDataType="SYNTHEA" --datastore-name "Test_Data_Store" --tags '[{"Key": "key1",
 "Value": "value1"}, {"Key": "key2", "Value": "value2"}]' --region us-east-1
```

To add tags to an existing Data Store, you would run the following example command:

```
aws healthlake tag-resource --resource-arn "arn:aws:healthlake:us-
east-1:691207106566:datastore/fhir/0725c83f4307f263e16fd56b6d8ebdbe" --tags '[{"Key":
 "key1", "Value": "value1"}]' --region us-east-1
```

If successful, this command returns no response.

Listing tags for a Data Store

Follow these steps to use the AWS CLI to view a list of the AWS tags for a HealthLake Data Store. If no tags have been added, the returned list is empty.

At the terminal or command line, run the list-tags-for-resource command as shown in the following example.

```
aws healthlake-test list-tags-for-resource --resource-arn "arn:aws:healthlake:us-
east-1:674914422125:datastore/fhir/0725c83f4307f263e16fd56b6d8ebdbe" --region us-east-1
```

```
{
    "tags": {
        "key": "value",
        "key1": "value1"
    }
}
```

Removing tags from a Data Store

You can remove one or more tags associated with a Data Store. Removing a tag does not delete the tag from other AWS resources that are associated with that tag.

At the terminal or command line, run the untag-resource command, specifying the Amazon Resource Name (ARN) of the Data Store where you want to remove tags and the tag key of the tag you want to remove.

```
aws healthlake untag-resource --resource-arn "arn:aws:healthlake:us-
east-1:674914422125:datastore/fhir/b91723d65c6fdeb1d26543a49d2ed1fa" --tag-keys '["key1"]'
 --region us-east-1
```

If successful, this command does not return a response. To verify the tags associated with the Data Store, run the list-tags-for-resource command.

Monitoring *HealthLake*

Monitoring is an important part of maintaining the reliability, availability, and performance of *HealthLake* and your other AWS solutions. AWS provides the following monitoring tools to watch *HealthLake*, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a specific threshold . For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account. It then delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address for those calls, and when they occurred. For more information, see the AWS CloudTrail User Guide.

Monitoring *HealthLake* with Amazon CloudWatch

You can monitor *HealthLake* using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so you can use that historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

Metrics are reported for all HealthLake APIs, including the following.

- Data Store Management APIs —CreateFHIRDatastore, DeleteFHIRDatastore, DescribeFHIRDatastore, ListFHIRDatastores
- Import and Export APIs —StartFHIRImportJob, ListFHIRImportJobs, DescribeFHIRImportJob, StartFHIRExportJob, ListFHIRExportJobs, DescribeFHIRExportJob
- HTTP REST Client and resource management APIs — CreateResource, DeleteResource, GetCapabilities, GetFullHistory, GetHistoryByResourceId, GetHistoryByResourceType, ReadResource, SearchAll, SearchWithGet, SearchWithPost, Transaction, UpdateResource, ValidateExistingResource, ValidateResource, VersionReadResource.
- Tagging APIs — ListTagsForResource, TagResource, UntagResource

The following tables list the metrics and dimensions for *HealthLake*.

The following metrics are reported. Each is presented as a frequency count for a user specified data range.

Metrics

Metrics	Description
Call Count	The number of calls to APIs. This can be reported either for the account or a specified Data Store. Units: Count

Metrics	Description
	Valid Statistics: Sum, Count Dimensions: Operation, Data Store ID, Data Store type
Successful Requests	The number of successful API requests. Units: Count Valid Statistics: Sum, Average Dimensions: Operation, Data Store ID, Data Store type
User Errors	The number of requests that failed due to user error. Units: Count Valid Statistics: Sum, Average Dimensions: Operation, Data Store ID, Data Store type
Server Errors	The number of requests that failed due to server error. Units: Count Valid Statistics: Sum, Average Dimensions: Operation, Data Store ID, Data Store type
Throttled Requests	The number of requests that have been throttled. This metric is not included in user or server errors counts. Units: Count Valid Statistics: Sum, Average Dimensions: Operation, Data Store ID, Data Store type
Latency	The time it took in milliseconds to process the user request. Unit: Milliseconds Valid statistics: Minimum, Maximum, Average Dimensions: Operation, Data Store ID, Data Store type

The following dimensions are reported.

Dimensions

Dimensions	Description
Operation	Which API operation was used
DataStoreID	The Data Store included in the API request
DataStoreType	The type of Data Store (currently only FHIR R4 is supported)

You can get metrics for HealthLake with the AWS Management Console, the AWS CLI, or the CloudWatch API. You can use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools. The HealthLake console displays graphs based on the raw data from the CloudWatch API.

You must have the appropriate CloudWatch permissions to monitor HealthLake with CloudWatch. For more information, see Authentication and Access Control for Amazon CloudWatch in the Amazon CloudWatch User Guide.

Viewing *HealthLake* metrics

To view metrics (HealthLake console)

1. Sign in to the AWS Management Console and open the HealthLake console.
2. From the list of Data Stores, choose the one whose metrics you want to see.
3. Choose **Monitoring**. Metrics are displayed in graphs.

To view metrics (CloudWatch console)

1. Sign in to the AWS Management Console and open the CloudWatch console.
2. Choose **Metrics**, choose **All Metrics**, and then choose **AWS/HealthLake**.
3. Choose the dimension, choose a metric name, then choose **Add to graph**.
4. Choose a value for the date range. The metric count for the selected date range is displayed in the graph.

Creating an alarm using CloudWatch

A CloudWatch alarm watches a single metric over a specified time period, and performs one or more actions: sending a notification to an Amazon Simple Notification Service (Amazon SNS) topic or Auto Scaling policy. The action or actions are based on the value of the metric relative to a given threshold over a number of time periods that you specify. CloudWatch can also send you an Amazon SNS message when the alarm changes state.

CloudWatch alarms invoke actions only when the state changes and has persisted for the period you specify.

To view metrics (CloudWatch console)

1. Sign in to the AWS Management Console and open the CloudWatch console.
2. Choose **Alarms**, and then choose **Create Alarm**.
3. Choose **AWS/HealthLake**, and then choose a metric.
4. For **Time Range**, choose a time range to monitor, and then choose **Next**.

5. Enter a **Name** and **Description**.
6. For Whenever, choose **>=**, and type a maximum value.
7. If you want CloudWatch to send an email when the alarm state is reached, in the Actions section, for Whenever this alarm, choose State is **ALARM**. For Send notification to, choose a mailing list or choose **New list** and create a new mailing list.
8. Preview the alarm in the Alarm Preview section. If you are satisfied with the alarm, choose **Create Alarm**.

Creating and monitoring FHIR Data Stores in Amazon HealthLake

Important notice

Amazon HealthLake protects customer data under the AWS Shared Responsibility Model policies. Remember that customer-inputed names should never contain personally identifiable information (PII) or protected health information (PHI). For more information, see the HealthLake Security chapter.

Creating and monitoring FHIR resources

With Amazon HealthLake, you can create and manage Data Stores for storing FHIR resources. Use the following operations to create and monitor the Data Store:

- **CreateFHIRDataStore** – Creates a Data Store that can ingest and export FHIR data.
- **DescribeFHIRDataStore** – Gets the properties associated with the FHIR datastore, including Data Store ID, Data Store Arn, Data Store name, Data Store status, created at, Data Store type version, and Data Store endpoint. These items are used in subsequent operations.
- **ListFHIRDataStores** – Lists all FHIR Data Stores that are in the user's account, regardless of Data Store status.
- **DeleteFHIRDataStores** – Deletes a FHIR Data Store and all of the data included within.

Creating a Data Store example

The following example demonstrates how to create a Data Store using the AWS Command Line Interface You can also use either the console or the create-fhir-datastore API to create a Data Store. When you create your Data Store, encryption at rest defaults to an AWS-owned KMS key unless specified otherwise.

```
   aws healthlake create-fhir-datastore \
   --datastore-type-version R4 \
   --sse-configuration '{"KmsEncryptionConfig":
{"CmkType":"CUSTOMER_MANAGED_KMS_KEY","KmsKeyId":"KMS_KEY_ID_OR_ARN_OR_ALIAS"}}'
   --preload-data-config PreloadDataType="SYNTHEA" \
   --datastore-name "FhirTestDatastore"
```

The JSON response includes the Data Store ID, Amazon Resource Name (ARN), and confirmation that the Data Store is being created. When the Data Store is ready to ingest data, the status is `"ACTIVE"`. The following is the example JSON response.

```
{
    "DatastoreEndpoint": "https://healthlake.us-east-1.amazonaws.com/datastore/(Datastore
 ID)/r4/",
```

```
    "DatastoreArn": "arn:aws:healthlake:us-east-1:(AWS Account ID):datastore/(Datastore
 ID)",
    "DatastoreStatus": "CREATING",
    "DatastoreId": "(Datastore ID)"
}
```

Describing a Data Store example

To learn the properties of a Data Store, use the describe-fhir-datastore API, the console, or the AWS CLI command as shown in the following example.

```
aws healthlake describe-fhir-datastore \
    --datastore-id "1f2f459836ac6c513ce899f9e4f66a59" \
    --region us-east-1
```

In the JSON response, you can learn the `DatastoreName` , `DatastoreArn(ARN)`, `DatastoreEndpoint`, `DatastoreStatus`, `DatastoreTypeVersion`, and `DatastoreId`, as shown in the following example.

```
{
    "DatastoreProperties": {
        "PreloadDataConfig": {
            "PreloadDataType": "SYNTHEA"
        },
        "DatastoreName": "FhirTestDatastore",
        "DatastoreArn": "arn:aws:healthlake:us-east-1:(AWS Account ID):datastore/(Datastore
 ID)",
        "DatastoreEndpoint": "https://healthlake.us-east-1.amazonaws.com/datastore/
(Datastore ID)/r4/",
        "DatastoreStatus": "CREATING",
        "DatastoreTypeVersion": "R4",
        "DatastoreId": "(Datastore ID)"
    }
}
```

Listing Data Stores example

Use the list-fhir-datastore API or the console to find the names, properties, and statuses of the Data Stores associated with your account as shown in the following example. You can also set filters to focus your listings to `'ACTIVE'` Data Stores only, as shown in the example.

```
aws healthlake list-fhir-datastores
    --region us-east-1
    --filter DatastoreStatus=ACTIVE
```

The following is the response in JSON.

```
{
     "DatastorePropertiesList": [
    {
        "PreloadDataConfig": {
            "PreloadDataType": "SYNTHEA"
        },
```

```
        "DatastoreName": "FhirTestDatastore",
        "DatastoreArn": "arn:aws:healthlake:us-east-1:(AWS Account ID):datastore/(Datastore
 ID)",
        "DatastoreEndpoint": "https://healthlake.us-east-1.amazonaws.com/datastore/
(Datastore ID)/r4/",
        "DatastoreStatus": "ACTIVE",
        "DatastoreTypeVersion": "R4",
        "CreatedAt": 1605574003.209,
        "DatastoreId": "(Datastore ID)"
    },
    {
        "DatastoreName": "Demo",
        "DatastoreArn": "arn:aws:healthlake:us-east-1:(AWS Account ID):datastore/(Datastore
 ID)",
        "DatastoreEndpoint": "https://healthlake.us-east-1.amazonaws.com/datastore/
(Datastore ID)/r4/",
        "DatastoreStatus": "ACTIVE",
        "DatastoreTypeVersion": "R4",
        "CreatedAt": 1603761064.881,
        "DatastoreId": "(Datastore ID)"
    }
    ]
}
```

Deleting a Data Store example

To delete a data store and all information in it, use the `DeleteFHIRDataStore` command using the AWS CLI as shown in the following example. You can also delete a data store using the delete-fhir-datastore API or the console. Deleting a Data Store removes all of the FHIR resource versions contained within the Data Store and the underlying infrastructure. Logs related to a deleted Data Store are retained within the service account in accordance with HIPAA guidelines.

Deleting a Data Store is an asynchronous operation. When you start deleting a Data Store, the Data Store status changes to `DELETING`. It remains in the `DELETING` state until all of the FHIR data from the Data Store is removed along with the underlying infrastructure necessary to support the data. Once the data and infrastructure are removed, the status of the Data Store changes to `DELETED`. After deletion, the Data Store appears in the `DescribeFHIRDataStore` and `ListFHIRDataStores` operations for seven days. After seven days the Data Store will no longer appear in the results.

```
aws healthlake delete-fhir-datastore
    --datastore-id (Data Store ID)
```

As shown in the following example JSON response, the status changes to "`DELETING`" to confirm that the Data Store and its contents are in the process of being deleted.

```
{
    "DatastoreEndpoint": "https://healthlake.us-east-1.amazonaws.com/datastore/(Datastore
 ID)/r4/",
    "DatastoreArn": "arn:aws:healthlake:us-east-1:(AWS Account ID):datastore/(Datastore
 ID)",
    "DatastoreStatus": "DELETING",
    "DatastoreId": "(Datastore ID)"
}
```

Managing FHIR resources in Amazon HealthLake

Amazon HealthLake enables users to manage their FHIR resources in a Data Store through an HTTP client or the console.

Managing FHIR resources

Use the Amazon HealthLake FHIR REST APIs to can create, read, update, and delete resources in an active Data Store. To see all of the active Data Store's FHIR-related capabilities, see the capability statement for the active Data Store through either `GETCapabilities` in an HTTP client or the console. FHIR resources are validated against the FHIR R4 Structure Definition. If a resource is invalid, an OperationOutcome message will result with details explaining the exception.

The following table lists the actions that are supported by HealthLake.

Supported Operations

Operation	Description	Instance-level, Type-level or Whole-system interaction
Read	Read the current state of the resource	Instance-level
Update	Update a resource by its ID (or create it if it's new)	Instance-level
Delete	Delete a resource	Instance-level
Create	Create a new resource with a server-assigned ID	Type-level
Search	Search the resource type based on filter criteria	Type-level
Capabilities	Get a capability statement for the system	Whole-system level

The following table lists the resource types supported by HealthLake. You can use these resources as search criteria or as part of analysis workflows.

Supported FHIR Resource Types

Account	DocumentReference	Organization
ActivityDefinition	Encounter	Parameters
AllergyIntolerance	Endpoint	Patient
Appointment	EpisodeofCare	PaymentNotice

AppointmentResponse	ExplanationOfBenefit	PaymentReconciliation
AuditEvent	FamilyMemberHistory	Person
Binary	Goal	Practitioner
Bundle	GuidanceResponse	PractitionerRole
CapabilityStatement	HealthcareService	Procedure
CarePlan	ImagingStudy	Provenance
CareTeam	Immunization	Questionnaire
Claim	Library	QuestionnaireResponse
ClaimResponse	Location	RelatedPerson
CodeSystem	Measure	RequestGroup
Communication	MeasureReport	Schedule
CommunicationRequest	Medication	ServiceRequest
ConceptMap	MedicationAdministration	Slot
Condition	MedicationDispense	Specimen
Coverage	MedicationRequest	StructureDefinition
CoverageEligibilityRequest	MedicationStatement	StructureMap
CoverageEligibilityResponse	MessageHeader	Substance
Device	NutritionOrder	ValueSet
DiagnosticReport	Observation	VisionPrescription
DocumentManifest	OperationOutcome	

Note: An AuditEvent resource can be created or read, but can not be updated or deleted.

Example: Using Create with POST

The following example shows up to create a patient FHIR resource using `POST`.

```
POST /datastore/(Datastore ID)/r4/Patient/ HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential=(Redacted)

{
  "resourceType": "Patient",
  "active": true,
  "name": [
    {
      "use": "official",
      "family": "Doe",
      "given": [
```

```
        "Jane"
      ]
    },
    {
      "use": "usual",
      "given": [
        "Jane"
      ]
    }
  ],
  "gender": "female",
  "birthDate": "1966-09-01"
}
```

To confirm creation of the patient resource, you get the following JSON response.

```
{
  "resourceType": "Patient",
  "active": true,
  "name": [
    {
      "use": "official",
      "family": "Doe",
      "given": [
        "Jane"
      ]
    },
    {
      "use": "usual",
      "given": [
        "Jane"
      ]
    }
  ],
  "gender": "female",
  "birthDate": "1966-09-01"
}
```

Example: Reading a resource with GET

The following example shows how to read a patient FHIR resource using `GET` and using the logical ID (`LID`) to define the resource.

```
GET /datastore/(Datastore ID)/r4/Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9 HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential=(Redacted)
```

You receive the following JSON in response.

```
{
    "resourceType": "Patient",
```

```
    "active": true,
    "name": [
        {
            "use": "official",
            "family": "Doe",
            "given": [
                "Jane"
            ]
        },
        {
            "use": "usual",
            "given": [
                "Jane"
            ]
        }
    ],
    "gender": "female",
    "birthDate": "1966-09-01",
    "meta": {
        "lastUpdated": "2020-11-23T06:24:13.202Z"
    },
    "id": "2de04858-ba65-44c1-8af1-f2fe69a977d9"
}
```

Example: Updating a resource using PUT

The following example shows how to use `PUT` to update a patient FHIR resource to correct inaccurate patient information.

```
PUT /datastore/(Datastore ID)/r4/Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9 HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential=(Redacted)

{
  "id": "2de04858-ba65-44c1-8af1-f2fe69a977d9",
  "resourceType": "Patient",
  "active": true,
  "name": [
    {
      "use": "official",
      "family": "Doe",
      "given": [
        "Jane"
      ]
    },
    {
      "use": "usual",
      "given": [
        "Jane"
      ]
    }
  ],
  "gender": "female",
  "birthDate": "1985-12-31"
}
```

You receive the following JSON in response to confirm the change.

```
{
    "id": "2de04858-ba65-44c1-8af1-f2fe69a977d9",
    "resourceType": "Patient",
    "active": true,
    "name": [
        {
            "use": "official",
            "family": "Doe",
            "given": [
                "Jane"
            ]
        },
        {
            "use": "usual",
            "given": [
                "Jane"
            ]
        }
    ],
    "gender": "female",
    "birthDate": "1985-12-31",
    "meta": {
        "lastUpdated": "2020-11-23T06:43:45.133Z"
    }
    "id": "2de04858-ba65-44c1-8af1-f2fe69a977d9"
}
```

Example: Deleting a resource

The following example shows how to delete a patient FHIR resource.

```
DELETE /datastore/(Datastore ID)/r4/Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9 HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential=(Redacted)
```

You receive the following response, confirming that the resource is no longer in the Data Store.

```
204 No Content Response Code
```

Importing files to a FHIR Data Store

After your FHIR Data Store has been created, you can import files from an Amazon Simple Storage Service (Amazon S3) bucket. You can use the console to create and manage import jobs, or use the import APIs. Amazon HealthLake accepts input files in newline delimited JSON (.ndjson) format, where each line consists of a valid FHIR resource. The APIs start, describe, and list ongoing import jobs. A customer-owned or AWS-owned KMS key is required for encryption of the Amazon S3 bucket for all import jobs. To learn more about creating and using a KMS Keys, see Creating keys in the AWS Key Management Service developer guide.

Only one import or export job can run at time for an active Data Store. However, users can create, read, update, or delete FHIR resources while an import job is in progress.

For each import job, output files are generated for successess or failures that can be analyzed via the Manifest.json file. Users can programatically navigate to these output files, and they are organized into two folders named SUCCESS and FAILURE. An output file is generated for each file in the import job. Because the output files may contain sensitive information, users are required to provide both an output Amazon S3 bucket and a KMS key for encryption.

The following is an example of the output Manifest.json file. It is recommended users look at the file as the first step of troubleshooting a failed import job because it provides details on each file and what caused the import job to fail.

```
  {
  "inputDataConfig": {
    "s3Uri": "s3://inputS3Bucket/healthlake-input/invalidInput/"
  },
  "outputDataConfig": {
    "s3Uri": "s3://outputS3Bucket/32839038a2f47f17c2fe0f53f0c3a0ba-
FHIR_IMPORT-19dd7bb7bcc8ee12a09bf6d322744a3d/",
    "encryptionKeyID": "arn:aws:kms:us-west-2:123456789012:key/fbbbfee3-20b3-42a5-a99d-
c48c655ed545"
  },
  "successOutput": {
    "successOutputS3Uri": "s3://outputS3Bucket/32839038a2f47f17c2fe0f53f0c3a0ba-
FHIR_IMPORT-19dd7bb7bcc8ee12a09bf6d322744a3d/SUCCESS/"
  },
  "failureOutput": {
    "failureOutputS3Uri": "s3://outputS3Bucket/32839038a2f47f17c2fe0f53f0c3a0ba-
FHIR_IMPORT-19dd7bb7bcc8ee12a09bf6d322744a3d/FAILURE/"
  },
  "numberOfScannedFiles": 1,
  "numberOfFilesImported": 1,
  "sizeOfScannedFilesInMB": 0.023627,
  "sizeOfDataImportedSuccessfullyInMB": 0.011232,
  "numberOfResourcesScanned": 9,
  "numberOfResourcesImportedSuccessfully": 4,
  "numberOfResourcesWithCustomerError": 5,
  "numberOfResourcesWithServerError": 0
}
```

Performing an import

You can start an import job using either the Amazon HealthLake console or the Amazon HealthLake import API, start-fhir-import-job API.

Importing files using the APIs

Prerequisites

When you use the Amazon HealthLake APIs, you must first create an AWS Identity Access and Management (IAM) policy and attach it to an IAM role. To learn more about IAM roles and trust policies, see IAM Policies and Permissions. Customers must also use a KMS key for encryption. To learn more about using KMS Keys, see Amazon Key Management Service.

To import files (API)

1. Upload your data into an Amazon S3 bucket.
2. To start a new import job, use the `start-FHIR-import-job` operation. When you start the job, tell HealthLake the name of the Amazon S3 bucket that contains the input files, the KMS key you wish to use for encryption, and the output data configuration.
3. To learn more about a FHIR import job, use the describe-fhir-import-job operation to get the job's ID, ARN, name, start time, end time, and current status. Use list-fhir-import-job to show all import jobs and their statuses.

Importing files using the console

To import files (console)

1. Upload your data into an Amazon S3 bucket.
2. To start a new import job, identify the Amazon S3 bucket and either create or identify the IAM role and the KMS key you want to use. To learn more about IAM roles and trust policies, see IAM Roles. To learn more about using KMS Keys, see Amazon Key Management Service.
3. Monitor the status of your job. The console shows the status of all imports that are in progress.

IAM policies for import jobs

The IAM role that calls the Amazon HealthLake APIs must have a policy that grants access to the Amazon S3 buckets containing the input files. It must also be assigned a trust relationship that enables HealthLake to assume the role. To learn more about IAM roles and trust policies, see IAM Roles.

The role must have the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketPublicAccessBlock",
                "s3:GetEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::inputS3Bucket",
                "arn:aws:s3:::outputS3Bucket"
            ],
            "Effect": "Allow"
        },
        {
```

```
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputS3Bucket/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::outputS3Bucket/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey*"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:012345678910:key/d330e7fc-b56c-4216-a250-
f4c43ef46e83"
            ],
            "Effect": "Allow"
        }
    ]
}
```

The role must have the following trust relationship.

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

Example: Starting and monitoring import jobs using the AWS CLI

The following example shows how to use the AWS CLI to start and monitor an import job. You can also use the start-fhir-import-job API or the console.

```
aws healthlake start-fhir-import-job \
--input-data-config S3Uri=s3://inputS3Bucket/inputFolder/ \
```

```
--datastore-id (Datastore ID) \
--data-access-role-arn "arn:aws:iam::012345678910:role/DataAccessRole" \
--job-output-data-config '{"S3Configuration": {"S3Uri":"s3://outputS3Bucket/healthlake-
output","KmsKeyId":"arn:aws:kms:us-east-1:012345678910:key/d330e7fc-b56c-4216-a250-
f4c43ef46e83"}}' \
--region us-east-1
```

When the import job begins, you'll receive the following confirmation.

```
{
 "JobId": "8a4077553e9a485ad889c1a89c7541f0",
 "JobStatus": "SUBMITTED",
 "DatastoreId": "32839038a2f47f17c2fe0f53f0c3a0ba"
}
```

To monitor the status of an import job or to learn its configuration properties, use the describe-fhir-import-job API or the AWS CLI command as shown in the following example.

```
    aws healthlake describe-fhir-import-job \
    --datastore-id (Datastore ID) \
    --job-id c145fbb27b192af392f8ce6e7838e34f \
    --region us-east-1
```

You receive the following information in response.

```
{
    "ImportJobProperties": {
    "InputDataConfig": {
    "S3Uri": "s3://(Bucket Name)/(Prefix Name)/"
    },
    "DataAccessRoleArn": "arn:aws:iam::(AWS Account ID):role/(Role Name)",
    "JobStatus": "COMPLETED",
    "JobId": "c145fbb27b192af392f8ce6e7838e34f",
    "SubmitTime": 1606272542.161,
    "EndTime": 1606272609.497,
    "DatastoreId": "(Datastore ID)"
    }
}
```

To see a list of all import jobs, use the list-fhir-import-jobs API or the AWS CLI command as shown in the following example. Users can add one or more filters to limit the results as shown.

```
    aws healthlake list-fhir-import-jobs\
    --datastore-id (Datastore ID) \
    --submitted-before (DATE like 2024-10-13T19:00:00Z)\
    --submitted-after (DATE like 2020-10-13T19:00:00Z )\
    --job-name "FHIR-IMPORT" \
    --job-status SUBMITTED  \
    --max-results (Integer between 1 and 500)
```

You receive the following information in response.

```
{
```

```
    "ImportJobProperties": {
    "OutputDataConfig": {
    "S3Uri": "s3://(Bucket Name)/(Prefix Name)/",
    "S3Configuration": {
     "S3Uri": "s3://(Bucket Name)/(Prefix Name)/",
     "KmsKeyId" : "(KmsKey Id)"
     },
    },
    "DataAccessRoleArn": "arn:aws:iam::(AWS Account ID):role/(Role Name)",
    "JobStatus": "COMPLETED",
    "JobId": "c145fbb27b192af392f8ce6e7838e34f",
    "JobName" "FHIR-IMPORT",
    "SubmitTime": 1606272542.161,
    "EndTime": 1606272609.497,
    "DatastoreId": "(Datastore ID)"
    }
}
"NextToken": String
```

Exporting files from a FHIR Data Store

After your data is imported to a Data Store, you can export files to an Amazon Simple Storage Service (Amazon S3) bucket for analysis or downstream applications using other AWS services. Amazon HealthLake exports files in newline delimited JSON (.ndjson) format, where each line consists of a valid FHIR resource. A customer-owned KMS key is required for encryption of the Amazon S3 bucket for all export jobs. To learn more about creating a KMS key, see Creating keys in the AWS Key Management Service developer guide.

Only one import or export job can run at time for an active Data Store. However, users can create, read, update, or delete FHIR resources while an export job is in progress.

Performing an export

You can start an export job using either the Amazon HealthLake console or the Amazon HealthLake export API, start-fhir-export-job API.

Exporting from your Data Store

Prerequisites

To use Amazon HealthLake APIs, you must create an AWS Identity Access and Management (IAM) policy and attach it to an IAM role. To learn more about IAM roles and trust policies, see IAM Policies and Permissions.

To export files

1. Create an S3 bucket. The Amazon S3 bucket must be in the same AWS region as the service, and BlockPublicAccess must be turned on for all options. To learn more, see Using Amazon S3 block public access. An Amazon-owned or customer-owned KMS key must also be used for encryption. To learn more about using KMS keys, see Amazon Key Management Service.
2. Create and add an IAM policy to allow the user to create and attach roles and policies. The following is an example.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
         "Action": [
            "iam:CreateRole",
            "iam:CreatePolicy",
            "iam:AttachRolePolicy",
            "iam:PassRole",
            "healthlake:*"
         ],
         "Effect": "Allow",
         "Resource": "*"
```

```
      }
   ]
}
```

3. Create a data access role. HealthLake uses this to write the output Amazon S3 bucket.
4. Add a trust policy to the data access role. The following is an example trust policy.

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

5. Add a permission policy to the data access role that enables the role to access the S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketPublicAccessBlock",
                "s3:GetEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::outputS3Bucket"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::outputS3Bucket/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey*"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:012345678910:key/d330e7fc-b56c-4216-a250-
f4c43ef46e83"
            ],
            "Effect": "Allow"
        }
    ]
}
```

6. Use the start-fhir-export-job operation to begin a bulk export job.

7. To get the ID, ARN, name, start time, end time, and current status of a FHIR export job, use describe-fhir-export-job. Use list-fhir-export-jobs to list all export jobs and their statuses.

Exporting files (console)

To export files (console)

1. Create an output S3 bucket in the same region as HealthLake.
2. To start a new export job, identify the output Amazon S3 bucket and either create or identify the IAM role that you want to use. To learn more about IAM roles and trust policies, see IAM Roles. An Amazon-owned or customer-owned KMS key must also be used for encyrption. To learn more about using KMS keys, see Amazon Key Management Service.
3. Monitor the status of your job. The console shows the status of all exports that are in progress.

Example: Starting export jobs

The following example shows how to use the AWS CLI to start and monitor an export job. You can also use the start-fhir-export-job API or the console.

```
aws healthlake start-fhir-export-job \
--datastore-id b35525cfbd0672ed444bd191aeaa558e \
--data-access-role-arn "arn:aws:iam::012345678910:role/Dummy" \
--output-data-config '{"S3Configuration": {"S3Uri":"s3://outputS3Bucket/healthlake-
output","KmsKeyId":"arn:aws:kms:us-east-1:012345678910:key/d330e7fc-b56c-4216-a250-
f4c43ef46e83"}}' \
--region us-east-1
```

You receive the following confirmation that the export job has begun.

```
{
    "DatastoreId": "(Datastore ID)",
    "JobStatus": "SUBMITTED",
    "JobId": "9b9a51943afaedd0a8c0c26c49135a31"
}
```

To monitor the status of an export job or to check its configuration properties, use the describe-fhir-export-job API or the AWS CLI command as shown in the following example.

```
aws healthlake describe-fhir-export-job \
    --datastore-id (Datastore ID) \
    --job-id 9b9a51943afaedd0a8c0c26c49135a31
```

You receive the following information in response.

```
{
    "ExportJobProperties": {
        "DataAccessRoleArn": "arn:aws:iam::(AWS Account ID):role/(Role Name)",
        "JobStatus": "IN_PROGRESS",
        "JobId": "9009813e9d69ba7cf79bcb3468780f16",
```

```
        "SubmitTime": 1609175692.715,
        "OutputDataConfig": {
            "S3Uri": "s3://(Bucket Name)/(Prefix Name)/59593b2d0367ce252b5e66bf5fd6b574-
FHIR_EXPORT-9009813e9d69ba7cf79bcb3468780f16/"
        },
        "DatastoreId": "(Datastore ID)"
    }
}
```

To see a list of all export jobs, use the list-fhir-export-jobs API or the AWS CLI command as shown in the following example.

```
    aws healthlake list-fhir-export-jobs\
    --datastore-id (Datastore ID) \
    --submitted-before (DATE like 2024-10-13T19:00:00Z)\
    --submitted-after (DATE like 2020-10-13T19:00:00Z )\
    --job-name "FHIR-EXPORT" \
    --job-status SUBMITTED  \
    --max-results (Integer between 1 and 500)
```

You receive the following information in response.

```
{
    "ExportJobProperties": {
    "OutputDataConfig": {
    "S3Uri": "s3://(Bucket Name)/(Prefix Name)/",
    "S3Configuration": {
     "S3Uri": "s3://(Bucket Name)/(Prefix Name)/",
     "KmsKeyId" : "(KmsKey Id)"
     },
    },
    "DataAccessRoleArn": "arn:aws:iam::(AWS Account ID):role/(Role Name)",
    "JobStatus": "COMPLETED",
    "JobId": "c145fbb27b192af392f8ce6e7838e34f",
    "JobName" "FHIR-EXPORT",
    "SubmitTime": 1606272542.161,
    "EndTime": 1606272609.497,
    "DatastoreId": "(Datastore ID)"
    }
}
"NextToken": String
```

Using FHIR search functionality in Amazon HealthLake

Amazon HealthLake provides the basic FHIR search functionality for querying a Data Store based on parameters or resource IDs to find specific records.

Using FHIR search in Amazon HealthLake

You can search for a specific resource type with supported search parameters. You can also search for resource IDs in the server without specifying the resource type. For more information about FHIR-supported searches, see the FHIR Search Documentation. HealthLake supports both Search with POST and Search with GET. Search with POST with the parameters in the request body is strongly recommended for all queries involving PHI and PII.

When results are paginated, all follow-up page requests will be Search with GET by default when the pagination token is used.

The following parameters are supported in HealthLake.

- **Number** – Searches for a numerical value.
- **Date/DateTime** – Searches for a date or time reference.
- **String** – Searches for a sequence of characters.
- **Token** – Searches for a close-to-exact match against a string of characters, often against a pair of values.
- **Composite** – Searches for multiple parameters for a single resource type, using the `AND` operation.
- **Quantity** – Searches for a number, system, and code as values. A number is required, but system and code are optional.
- **Reference** – Searches for references to other FHIR resources. An example is a search for a reference to a patient within an observation resource.
- **URI** – Searches for a string of characters that unambiguously identifies a particular resource.
- **Special** – Searches based on integrated medical NLP extensions.

To see which search parameters are supported for each resource type, use `GETCapabilities` in your FHIR HTTP client.

Supported search parameters and search modifiers in HealthLake

Common search parameters are supported for all FHIR resources, regardless of type. The following tables provide details about supported search parameter types, modifiers, comparators, common search parameters, and search operations.

Supported search parameters

Search parameter	Description
_id	Resource id (not a full URL)

Search parameter	Description
_lastUpdated	Date last updated. Server has discretion on the boundary precision.
_tag	Search by a resource tag.
_profile	Search for all resources tagged with a profile.

Search modifiers allow for different types of matching logic on string-based fields. For example, you can specify that a larger string should contain a smaller string, or you can specify that the string should be an exact match of the string that is passed in.

Supported search modifiers

Search modifier	Type
:missing	All parameters except `Composite`
:exact	String
:contains	String
:not	Token
:text	Token
:identifier	Reference

Search comparators allow you to control the nature of the matching for number, date, and quantity fields. The following table lists search comparators and their definitions.

Supported search comparators

Search comparator	Description
eq	The value for the parameter in the resource is equal to the provided value.
ne	The value for the parameter in the resource is not equal to the provided value.
gt	The value for the parameter in the resource is greater than the provided value.
lt	The value for the parameter in the resource is less than the provided value.
ge	The value for the parameter in the resource is greater or equal to the provided value.
le	The value for the parameter in the resource is less or equal to the provided value.
sa	The value for the parameter in the resource starts after the provided value.
eb	The value for the parameter in the resource ends before the provided value

Managing invalid search parameters

HealthLake manages invalid search parameters based on FHIR guidance. To specify how the server should handle unknown or unsupported search parameters, define the handling parameter in the HTTP header as follows.

- **handling = strict** – The server should return an error for unknown or unsupported search parameters.
- **handling = lenient** – The server should ignore unknown or unsupported search parameters.

Search with GET

HealthLake supports queries using Search with GET, however it is recommened that any queries involving PHI or PII are performed using Search with POST with the query parameters in the request body, not the URI. Parameters in Search with GET are only accepted with search parameters in the URI. The following example shows how to search `DocumentReferences` for streptococcal diagnosis and amoxicillin medication using an HTTP client with a `GET` search for the following input in HTTP.

```
GET /datastore/(Datastore ID)/r4/DocumentReference?_lastUpdated=le2021-12-19&infer-
icd10cm-entity-text-concept-score;=streptococcal|0.6&infer-rxnorm-entity-text-concept-
score=Amoxicillin|0.8 HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential= (Redacted)
```

You will receive the following JSON response.

```
{
    "resourceType": "Bundle",
    "type": "searchset",
    "entry": [
    {
       "resource": {
       "resourceType": "DocumentReference",
       "id": "985c3e94-4219-4c79-97a1-c94694525e24",
       "meta": {
          "lastUpdated": "2020-11-23T06:09:10.719Z"
        },
        "extension": [
          {
            "url": "http://healthlake.amazonaws.com/aws-cm/",
            "extension": [
              {
                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/",
                "extension": [
                  {
                    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/raw-
response",
                    "valueString": "{Entities: [{Id: 0,Text: otitis media,Category:
 MEDICAL_CONDITION,Type: DX_NAME,Score: 0.9815994,BeginOffset: 151,EndOffset:
 163,Attributes: [],Traits: [{Name: DIAGNOSIS,Score: 0.95042425}],ICD10CMConcepts:
 [{Description: Otitis media, unspecified, unspecified ear,Code: H66.90,Score: 0.7176407},
 {Description: Otitis media, unspecified,Code: H66.9,Score: 0.6930445}, {Description:
 Otitis media, unspecified, left ear,Code: H66.92,Score: 0.688161}, {Description: Otitis
 media, unspecified, bilateral,Code: H66.93,Score: 0.6748094}, {Description: Otitis media,
 unspecified, right ear,Code: H66.91,Score: 0.6645618}]}, {Id: 1,Text: streptococcal
 sore throat,Category: MEDICAL_CONDITION,Type: DX_NAME,Score: 0.92208487,BeginOffset:
 461,EndOffset: 486,Attributes: [],Traits: [],ICD10CMConcepts: [{Description: Streptococcal
```

```
 pharyngitis,Code: J02.0,Score: 0.55638546}, {Description: Acute streptococcal
 tonsillitis, unspecified,Code: J03.00,Score: 0.53159785}, {Description: Streptococcal
 sepsis, unspecified,Code: A40.9,Score: 0.51865804}, {Description: Acute pharyngitis,
 unspecified,Code: J02.9,Score: 0.45085955}, {Description: Streptococcal infection,
 unspecified site,Code: A49.1,Score: 0.41550553}]}, {Id: 3,Text: disorder,Category:
 MEDICAL_CONDITION,Type: DX_NAME,Score: 0.9191257,BeginOffset: 488,EndOffset:
 496,Attributes: [],Traits: [{Name: DIAGNOSIS,Score: 0.93372077}],ICD10CMConcepts:
 [{Description: Parkinson's disease,Code: G20,Score: 0.6959145}, {Description:
 Illness, unspecified,Code: R69,Score: 0.68428487}, {Description: Disorder of bone,
 unspecified,Code: M89.9,Score: 0.6542605}, {Description: Unspecified mental disorder
 due to known physiological condition,Code: F09,Score: 0.6240179}, {Description: Mental
 disorder, not otherwise specified,Code: F99,Score: 0.61046}]}],ModelVersion: 0.1.0}"
                  },
                  {
                    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/model-
version",
                    "valueString": "0.1.0"
                  },
                  {
                    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-
icd10-entity",
                    "extension": [
                      {
                        "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-
icd10-entity-id",
                        "valueInteger": 0
                      },
                      {
                        "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-
icd10-entity-text",
                        "valueString": "otitis media"
                      },
                      {
                        "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-
icd10-entity-begin-offset",
                        "valueInteger": 151
                      },
                      {
                        "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-
icd10-entity-end-offset",
                        "valueInteger": 163
                      },
                      {
                        "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-
icd10-entity-score",
                        "valueDecimal": 0.9815994
                      },
                      {
                        "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-
icd10-entity-ConceptList",
                        "extension": [
                          {
                            "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-
cm-icd10-entity-Concept",
                            "extension": [
                              {
                                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
aws-cm-icd10-entity-Concept-Code",
                                "valueString": "H66.90"
                              },
                              {
                                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
aws-cm-icd10-entity-Concept-Description",
                                "valueString": "Otitis media, unspecified, unspecified ear"
                              },
                              {
```

```
                                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
aws-cm-icd10-entity-Concept-Score",
                                "valueDecimal": 0.7176407
                              }
                            ]
                          },
                          {
                            "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-
cm-icd10-entity-Concept",
                            "extension": [
                              {
                                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
aws-cm-icd10-entity-Concept-Code",
                                "valueString": "H66.9"
                              },
                              {
                                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
aws-cm-icd10-entity-Concept-Description",
                                "valueString": "Otitis media, unspecified"
                              },
                              {
                                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
aws-cm-icd10-entity-Concept-Score",
                                "valueDecimal": 0.6930445
                              }
                            ]
                          },
                          {
                            "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-
cm-icd10-entity-Concept",
                            "extension": [
                              {
                                "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
aws-cm-icd10-entity-Concept-Code",
                                "valueString": "H66.92"
                              }
                            ]
                          }
```

Search with POST

You can also use Search with POST to query your Data Store through either parameters in the URI or in a request body. You can not have parameters in both the URI & and the request body. To protect PHI or PII, customers are strongly advised to put parameters in a body request, not the URI, for queries.

Search with POST example with parameters in URI

The following example demonstrates how to perform the same search using an HTTP client with a POST search with parameters in URI for the following input in HTTP.

```
POST /datastore/(Datastore ID)/r4/DocumentReference/_search?
_lastUpdated=le2021-12-19&infer-icd10cm-entity-text-concept-score;=streptococcal|0.6&infer-
rxnorm-entity-text-concept-score=Amoxicillin|0.8 HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential= (Redacted)
```

Search with POST example with parameters in body

The following example demonstrates how to perform the same search using an HTTP client with a POST search with parameters in body for the following input in HTTP. To use Search With Post with parameters in the request body, you must use content-type “application/ x-www-form-urlencoded”.

```
POST /datastore/(Datastore ID)/r4/DocumentReference/_search HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/ x-www-form-urlencoded
Authorization: AWS4-HMAC-SHA256 Credential= (Redacted)
_lastUpdated=le2021-12-19&infer-icd10cm-entity-text-concept-score;=streptococcal|0.6&infer-
rxnorm-entity-text-concept-score=Amoxicillin|0.8
```

Integrated medical natural language processing (NLP)

Amazon HealthLake automatically integrates with natural language processing (NLP) for the `DocumentReference` resource type. The integrated medical NLP output is provided as an extension to the existing `DocumentReference` resource. The integration involves reading the text data within the resource, and then calling the following integrated medical NLP operations: `DetectEntities-V2`, `InferICD10-CM`, and `InferRxNorm`. The response of each of the integrated medical NLP APIs is appended to the `DocumentReference` resource as an extension that is searchable. This enables users to identify patients through elements of their records that were previously buried within unstructured text. When you create a resource in HealthLake, the resource is updated with the response from the integrated medical NLP operations. These extensions follow the FHIR format for extensions with an identifying URL, and the respective value for the URL.

Important notice

The Amazon HealthLake data transformations feature (integrated medical NLP) is not a substitute for professional medical advice, diagnosis, or treatment. HealthLake integrated medical NLP provides confidence scores that indicate the level of confidence in the accuracy of the detected entities. Identify the right confidence threshold for your use case, and use high confidence thresholds in situations that require high accuracy. In certain use cases, results should be reviewed and verified by appropriately trained human reviewers.

Restrictions for integrated medical NLP

Integration with medical NLP is nearly seamless. However, there are two constraints:

- If the `DocumentReference` object is larger than the quota for the operation, HealthLake does not initiate the operation. For information on quotas, see Quotas for Amazon HealthLake (p. 74)
- If a Data Store is deleted before the operation is initiated, an update fails.

Search parameters

The following table lists the searchable attributes for integrated medical NLP.

Search parameters

Search parameters	Finds matches for
detectEntities-entity-category	Entity Category within the DetectEntities subextension within the AWS CM Extension
detectEntities-entity-text	Entity Text within the DetectEntities subextension within the AWS CM Extension
detectEntities-entity-type	Entity Type within the DetectEntities subextension within the AWS CM Extension

Search parameters	Finds matches for
detectEntities-entity-score	Entity Score within the DetectEntities subextension within the AWS CM Extension
infer-icd10cm-entity-text	Entity Text within the InferICD10CM subextension within the AWS CM Extension
infer-icd10cm-entity-score	Entity Score within the InferICD10CM subextension within the AWS CM Extension
infer-icd10cm-entity-concept-code	Entity Concept Code within the InferICD10CM subextension within the AWS CM Extension
infer-icd10cm-entity-concept-description	Entity Concept Description within the InferICD10CM subextension within the AWS CM Extension
infer-icd10cm-entity-concept-score	Entity Concept Score within the InferICD10CM subextension within the AWS CM Extension
infer-rxnorm-entity-score	Entity Score within the InferRxNorm subextension within the AWS CM Extension
infer-rxnorm-entity-text	Entity Text within the InferRxNorm subextension within the AWS CM Extension
infer-rxnorm-entity-concept-code	Entity Concept Code within the InferRxNorm subextension within the AWS CM Extension
infer-rxnorm-entity-concept-description	Entity Concept Description within the InferRxNorm subextension within the AWS CM Extension
infer-rxnorm-entity-concept-score	Entity Concept Score within the InferRxNorm subextension within the AWS CM Extension

To match the criteria where `EntityText` and `EntityCategory` are part of the same entity, HealthLake provides special search. The following table describes the special search parameters that are supported within HealthLake.

Search parameters

Search parameters	Matches returned
detectEntities-entity-text-category	If there is at least one entity in the DetectEntities subextention that matches both the entityText and entityCategory.
detectEntities-entity-type-score	If there is at least one entity in the DetectEntities subextention that matches both the entityType and entityScore.
detectEntities-entity-text-score	If there is at least one entity in the DetectEntities subextention that matches both the entityText and entityScore.
detectEntities-entity-text-type	If there is at least one entity in the DetectEntities subextention that matches both the entityText and entityType.

Search parameters	Matches returned
detectEntities-entity-category-score	If there is at least one entity that matches both the entityCategory and entityScore.
infer-icd10cm-entity-text-concept-code	If there is at least one entity in the InferICD10CM subextension that matches the entityText and there is at least one conceptCode for that entity that matches the code.
infer-icd10cm-entity-text-concept-score	If there is at least one entity in the InferICD10CM subextension that matches the entityText and there is at least one conceptScore for that entity that matches the score.
infer-icd10cm-entity-concept-description-concept-score	If there is at least one concept within the entity in the InferICD10CM subextension that matches the concept description and the conceptScore.
infer-rxnorm-entity-text-concept-code	If there is at least one entity in the InferRxNorm subextension that matches the entityText and there is at least one conceptCode for that entity that matches the code.
infer-rxnorm-entity-text-concept-score	If there is at least one entity in the InferRxNorm subextension that matches the entityText and there is at least one conceptScore for that entity that matches the score.
infer-rxnorm-entity-concept-description-concept-score	If there is at least one concept within the entity in the InferRxNorm subextension that matches the concept description and the conceptScore.

Integrated medical NLP enrichment

The following example shows how integrated medical NLP detects entities and adds information to the FHIR resource for enrichment. The initial record would read as follows.

```
POST /datastore/(Datastore ID)/r4/DocumentReference/ HTTP/1.1
Host: healthlake.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential=(Redacted)

{
  "resourceType": "DocumentReference",
  "text": {
          "status": "generated",
          "div": "(div xmlns=\"http://www.w3.org/1999/xhtml\">\n\n\t\t\t\t\t\t\t< href=
\"http://localhost:9556/svc/fhir/Binary/1e404af3-077f-4bee-b7a6-a9be97e1ce32\">Document:
 urn:oid:129.6.58.92.88336</a>undefined, created 24/12/2005\n\t\t\t\t\t</div"
        },
   "status": "current",
   "content": [
    {
      "attachment": {
        "contentType": "text/plain",
        "data":
 "CjIwMTktMTItMzEKCiMgQ2hpZWYgQ29tcGxhaW50Ck5vIGNvbXBsYWludHMuCgojIEhpc3Rvcnkgb2YgUHJlc2VudCBJbGxuZXNzC
```

```
      }
    }
    ],
        "category": [
          {
            "coding": [
              {
                "system": "http://hl7.org/fhir/us/core/CodeSystem/us-core-
documentreference-category",
                "code": "clinical-note",
                "display": "Clinical Note"
              }
            ]
          }
        ],
  "subject": {
    "reference": "Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9"
  }
}
```

```
{
    "resourceType": "DocumentReference",
    "text": {
        "status": "generated",
        "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">\n\n\t\t\t\t\t\t\t<a href=
\"http://localhost:9556/svc/fhir/Binary/1e404af3-077f-4bee-b7a6-a9be97e1ce32\">Document:
 urn:oid:129.6.58.92.88336</a>undefined, created 24/12/2005\n\t\t\t\t\t</div>"
    },
    "status": "current",
    "content": [
        {
            "attachment": {
                "contentType": "text/plain",
                "data":
 "CjIwMTktMTItMzEKCiMgQ2hpZWYgQ29tcGxhaW50Ck5vIGNvbXBsYWludHMuCgojIEhpc3Rvcnkgb2YgUHJlc2VudCBJbGxuZXNz
            }
        }
    ],
    "category": [
        {
            "coding": [
                {
                    "system": "http://hl7.org/fhir/us/core/CodeSystem/us-core-
documentreference-category",
                    "code": "clinical-note",
                    "display": "Clinical Note"
                }
            ]
        }
    ],
    "subject": {
        "reference": "Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9"
    },
    "meta": {
        "lastUpdated": "2020-11-23T06:56:11.580Z"
    },
    "id": "3a43969a-c690-4480-85e3-7e2c65c8cefb"
}
```

After enrichment, the resource will have the following information when read. This is a truncated record for clarity. Included in the response is whether the operation was successful or unsupported.


```
{
    "resourceType": "DocumentReference",
    "status": "current",
    "content": [
        {
            "attachment": {
                "contentType": "text/plain",
                "data":
 "CjIwMTktMTItMzEKCiMgQ2hpZWYgQ29tcGxhaW50Ck5vIGNvbXBsYWludHMuCgojIEhpc3Rvcnkgb2YgUHJlc2VudCBJbGxuZXNzC
            }
        }
    ],
    "category": [
        {
            "coding": [
                {
                    "system": "http://hl7.org/fhir/us/core/CodeSystem/us-core-
documentreference-category",
                    "code": "clinical-note",
                    "display": "Clinical Note"
                }
            ]
        }
    ],
    "subject": {
        "reference": "Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9"
    },
    "id": "f40365c6-d714-4ce7-b190-43d0d2047a10",
    "meta": {
        "lastUpdated": "2021-05-07T17:47:35.331Z"
    },
    "extension": [
        {
            "extension": [
                {
                    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/",
                    "extension": [
                        {
                            "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/raw-
response",
                            "valueString": "{Entities: [{Id: 1,Text: streptococcal
 sore throat,Category: MEDICAL_CONDITION,Type: DX_NAME,Score: 0.9721338,BeginOffset:
 151,EndOffset: 176,Attributes: [],Traits: [],ICD10CMConcepts: [{Description:
 Streptococcal pharyngitis,Code: J02.0,Score: 0.67831886}, {Description: Streptococcal
 sepsis, unspecified,Code: A40.9,Score: 0.6346688}, {Description: Acute pharyngitis,
 unspecified,Code: J02.9,Score: 0.6200017}, {Description: Acute streptococcal tonsillitis,
 unspecified,Code: J03.00,Score: 0.6014083}, {Description: Acute pharyngitis,Code:
 J02,Score: 0.55188143}]}, {Id: 3,Text: disorder,Category: MEDICAL_CONDITION,Type:
 DX_NAME,Score: 0.8767418,BeginOffset: 178,EndOffset: 186,Attributes: [],Traits:
 [{Name: DIAGNOSIS,Score: 0.9286054}],ICD10CMConcepts: [{Description: Unspecified
 disorder of ear, unspecified ear,Code: H93.90,Score: 0.7463527}, {Description:
 Disorder of brain, unspecified,Code: G93.9,Score: 0.72858506}, {Description:
 Endocrine disorder, unspecified,Code: E34.9,Score: 0.7257518}, {Description: Disorder
 of thyroid, unspecified,Code: E07.9,Score: 0.72223496}, {Description: Personal
 history of other diseases of the nervous system and sense organs,Code: Z86.69,Score:
 0.68951124}]}, {Id: 4,Text: otitis media,Category: MEDICAL_CONDITION,Type: DX_NAME,Score:
 0.9772095,BeginOffset: 189,EndOffset: 201,Attributes: [],Traits: [{Name: DIAGNOSIS,Score:
 0.94999653}],ICD10CMConcepts: [{Description: Otitis media, unspecified, unspecified
 ear,Code: H66.90,Score: 0.7708893}, {Description: Otitis media, unspecified,
 bilateral,Code: H66.93,Score: 0.7557719}, {Description: Otitis media, unspecified,Code:
 H66.9,Score: 0.7519664}, {Description: Otitis media, unspecified, left ear,Code:
```

```
 H66.92,Score: 0.73516965}, {Description: Otitis media, unspecified, right ear,Code:
 H66.91,Score: 0.69751483}]}],ModelVersion: 0.1.0}"
                        },
                        {
                            "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/
model-version",
                            "valueString": "0.1.0"
                        },
                        {
                            "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-
cm-icd10-entity",
                            "extension": [
                                {
                                    "url": "http://healthlake.amazonaws.com/aws-cm/infer-
icd10/aws-cm-icd10-entity-id",
                                    "valueInteger": 1
                                },
...
                {
                    "url": "http://healthlake.amazonaws.com/aws-cm/status/",
                    "valueString": "SUCCESS"
                },
                {
                    "url": "http://healthlake.amazonaws.com/aws-cm/message/",
                    "valueString": "The Amazon HealthLake integrated medical NLP operation
 was successful."
                }
            ],
            "url": "http://healthlake.amazonaws.com/aws-cm/"
        }
    ]
}
```

Quotas for Amazon HealthLake

Throttling and quotas for Amazon HealthLake

The following table describes throttling limits for resource management within Amazon HealthLake for each customer account. For all operations, users will receive a `ThrottlingException` error message if throttling limits are exceeded.

A maximum quota of ten Data Stores are allowed per an account. For information about requesting a quota increase, see the console support center to create a case.

Description	Limit in Transactions per second(TPS) or requests per minute
CreateFHIRDatastore and DeleteFHIRDatastore	1 request per minute
DescribeFHIRDatastore	10 TPS
ListFHIRDatastores	10 TPS
CreateResource, ReadResource, DeleteResource	20 TPS
UpdateResource	100 TPS
GetCapabilities	10 TPS
SearchWithGet and SearchWithPost	100 TPS
StartFHIRImportJob and StartFHIRExportJob	1 request per minute, only 1 job permitted at a time
DescribeFHIRImportJob, DescribeFHIRExportJob, ListFHIRImportJobs, ListFHIRExportJobs	10 TPS
TagResource, UntagResource, ListTagsforResource	10 TPS
Maximum characters for a medical note within the DocumentReference ResourceType (CreateResource/UpdateResource)	40,000 characters

The following table lists the quotas for `Import` jobs. There is no limit on the number of patients or resources that can be ingested into a Data Store and there is no limit on the number of import jobs. However, the maximum size for each import job is 50 GB or 10,000 files. Typical resources are about 2 KB in size and each patient file has 500 resources on average. Based on these numbers, the resources for 50,000 patients can be imported in one import job.

Description	Limit
Maximum import job size	50 GB
Maximum on-disk size of an imported file	50 MB

Description	Limit
Maximum number of files	10,000
Supported file extension	'.ndjson'

Troubleshooting

The following documentation can help you troubleshoot problems you might have with your HealthLake Data Store.

Topics

Why can't I create a Data Store?

Data Store creation is dependent on IAM permissions and requires the use of a symmetrical customer-owned or Amazon-owned CMK. Make sure you have the correct permissions in your IAM policy.To learn more about KMS see Amazon Key Management Service.

Exceeded number of Data Stores allowed per account

HealthLake has a quota of ten Data Stores per account. To learn how to request a quota increase, visit AWS Support Center.

How do I create authorization for the FHIR RESTful APIs?

Users should use a Signature version 4 signing process to add authentication to HealthLake API requests sent through an HTTP client. To learn more, see Signature Version 4 signing process.

To create sigv4 authorization using the AWS SDK for Python, create a script similar to the following example.

```
import boto3
import requests
import json
from requests_auth_aws_sigv4 import AWSSigV4

# Set the input arguments
data_store_endpoint = 'https://healthlake.us-east-1.amazonaws.com/datastore/<datastore id>/
r4//'
```

```
resource_path = "Patient"
requestBody = {"resourceType": "Patient", "active": True, "name": [{"use":
 "official","family": "Dow","given": ["Jen"]},{"use": "usual","given": ["Jen"]}],"gender":
 "female","birthDate": "1966-09-01"}
region = 'us-east-1'

#Frame the resource endpoint
resource_endpoint = data_store_endpoint+resource_path
session = boto3.session.Session(region_name=region)
client = session.client("healthlake")

# Frame authorization
auth = AWSSigV4("healthlake", session=session)

# Calling data store FHIR endpoint using SigV4 auth

r = requests.post(resource_endpoint, json=requestBody, auth=auth, )
print(r.json())
```

Additional information about using sigv4 authorization using AWS SDK for Python can be found in the Boto3 credentials topic.

My data isn't in FHIR R4 format- can I still use HealthLake?

Only FHIR R4 formatted data can be imported into a HealthLake Data Store. For a list of partners who offer products to help users transform their data, see Amazon HealthLake Partners.

Why am I receiving AccessDenied Errors when using the FHIR RESTful APIs for a Data Store encrypted with a customer managed KMS key?

Permissions for both customer managed key and IAM policies are required for a user or role to access a Data Store. A user must have the required IAM permissions for using a customer managed key. If a user has revoked or retired a grant that gave HealthLake permission to use the customer managed KMS key, HealthLake will return an AccessDenied error.

HealthLake must have the permission in place to access customer data, to encrypt new FHR resources imported to a Data Store, and to decrypt the FHIR resources when they are requested.

To learn more, see Troubleshooting key access.

Why did my import fail?

A successful import job will generate a folder with output inputFileName.ndjson files, however individual records can fail to import. When this happens, a second FAILURE folder will be generated with a manifest of records that failed to be imported. The job output location to access the manifest file is JobProperties.JobOutputDataConfig.S3Configuration.S3Uri.

This manifest file contains details about the job output such as location of all successful responses (successOutput.successOutputS3Uri), the location of all failed responses (failureOutput.failureOutputS3Uri) and additional job metrics. The contents of manifest file are parsable programatically.

To analyze why an import job failed use the DescribeFHIRImportJob API to analyze the JobProperties. The following is recommended:

- If the status is FAILED and a message is present, the failures are related to job parameters such as input data size or number of input files being beyond HealthLake quotas.
- If the import job status is FAILED and a message is not present, go to the job output location to access the manifest file, Manifest.json.

For each input file, there is failure output file with input file name for any resource that fails to import.The responses contain line number (lineId) corresponding to the location of input data, FHIR response object (UpdateResourceResponse) and status code (statusCode) of the response.

A sample output file would look like this:


```
{"lineId":3, UpdateResourceResponse:{"jsonBlob":{"resourceType":"OperationOutcome","issue":
[{"severity":"error","code":"processing","diagnostics":"1 validation error detected: Value
 'Patient123' at 'resourceType' failed to satisfy constraint: Member must satisfy regular
 expression pattern: [A-Za-z]{1,256}"}]}, "statusCode":400}
{"lineId":5, UpdateResourceResponse:{"jsonBlob":{"resourceType":"OperationOutcome","issue":
[{"severity":"error","code":"processing","diagnostics":"This property must be an
 simple value, not a com.google.gson.JsonArray","location":["/EffectEvidenceSynthesis/
name"]},{"severity":"error","code":"processing","diagnostics":"Unrecognised
 property '@telecom'","location":["/EffectEvidenceSynthesis"]},
{"severity":"error","code":"processing","diagnostics":"Unrecognised
 property '@gender'","location":["/EffectEvidenceSynthesis"]},
{"severity":"error","code":"processing","diagnostics":"Unrecognised
 property '@birthDate'","location":["/EffectEvidenceSynthesis"]},
{"severity":"error","code":"processing","diagnostics":"Unrecognised
 property '@address'","location":["/EffectEvidenceSynthesis"]},
{"severity":"error","code":"processing","diagnostics":"Unrecognised
 property '@maritalStatus'","location":["/EffectEvidenceSynthesis"]},
{"severity":"error","code":"processing","diagnostics":"Unrecognised
 property '@multipleBirthBoolean'","location":["/EffectEvidenceSynthesis"]},
{"severity":"error","code":"processing","diagnostics":"Unrecognised
 property '@communication'","location":["/EffectEvidenceSynthesis"]},
{"severity":"warning","code":"processing","diagnostics":"Name should be usable
 as an identifier for the module by machine processing applications such as
 code generation [name.matches('[A-Z]([A-Za-z0-9_]){0,254}')]","location":
["EffectEvidenceSynthesis"]},{"severity":"error","code":"processing","diagnostics":"Profile
 http://hl7.org/fhir/StructureDefinition/EffectEvidenceSynthesis, Element
 'EffectEvidenceSynthesis.status': minimum required = 1, but only found 0","location":
["EffectEvidenceSynthesis"]},{"severity":"error","code":"processing","diagnostics":"Profile
 http://hl7.org/fhir/StructureDefinition/EffectEvidenceSynthesis, Element
 'EffectEvidenceSynthesis.population': minimum required = 1, but only found 0","location":
["EffectEvidenceSynthesis"]},{"severity":"error","code":"processing","diagnostics":"Profile
 http://hl7.org/fhir/StructureDefinition/EffectEvidenceSynthesis, Element
 'EffectEvidenceSynthesis.exposure': minimum required = 1, but only found 0","location":
["EffectEvidenceSynthesis"]},{"severity":"error","code":"processing","diagnostics":"Profile
 http://hl7.org/fhir/StructureDefinition/EffectEvidenceSynthesis,
 Element 'EffectEvidenceSynthesis.exposureAlternative': minimum required
 = 1, but only found 0","location":["EffectEvidenceSynthesis"]},
{"severity":"error","code":"processing","diagnostics":"Profile http://hl7.org/fhir/
StructureDefinition/EffectEvidenceSynthesis, Element 'EffectEvidenceSynthesis.outcome':
 minimum required = 1, but only found 0","location":["EffectEvidenceSynthesis"]},
{"severity":"information","code":"processing","diagnostics":"Unknown
 extension http://synthetichealth.github.io/synthea/disability-adjusted-
```

```
life-years","location":["EffectEvidenceSynthesis.extension[3]"]},
{"severity":"information","code":"processing","diagnostics":"Unknown extension
 http://synthetichealth.github.io/synthea/quality-adjusted-life-years","location":
["EffectEvidenceSynthesis.extension[4]"]}]}, "statusCode":400}
{"lineId":7, UpdateResourceResponse:{"jsonBlob":{"resourceType":"OperationOutcome","issue":
[{"severity":"error","code":"processing","diagnostics":"2 validation errors detected: Value
 at 'resourceId' failed to satisfy constraint: Member must satisfy regular expression
 pattern: [A-Za-z0-9-.]{1,64}; Value at 'resourceId' failed to satisfy constraint: Member
 must have length greater than or equal to 1"}]}, "statusCode":400}
{"lineId":9, UpdateResourceResponse:{"jsonBlob":{"resourceType":"OperationOutcome","issue":
[{"severity":"error","code":"processing","diagnostics":"Missing required id field in
 resource json"}]}, "statusCode":400}
{"lineId":15, UpdateResourceResponse:{"jsonBlob":
{"resourceType":"OperationOutcome","issue":
[{"severity":"error","code":"processing","diagnostics":"Invalid JSON found in input
 file"}]}, "statusCode":400}
```

The example shows that there were failures on line 3, 4, 7, 9, 15 from the corresponding input lines from input file. For each of those lines, the explanations are as follows:

- On Line 3, the response explains that resourceType provided in line 3 of input file is invalid.
- On Line 5, the response explains that there is a FHIR validation error in line 5 of input file.
- On Line 7, the response explains that there is a validation issue with resourceId provided as input.
- On Line 9, the response explains that input file must contain a valid resource id.
- On line 15, the response of input file is that the file is not in a valid JSON format.

Using Amazon HealthLake for healthcare analytics

For a full tutorial on how to create a Data Store, import your FHIR formatted data, and then generate a dashboard, see the Amazon HealthLake Work Shop.

To take your analysis further, you can use HealthLake with other AWS services, as demonstrated in the following blog post examples:

- Population health applications with Amazon HealthLake – Part 1: Analytics and monitoring using Amazon QuickSight.
- Building predictive disease models using Amazon SageMaker with Amazon HealthLake normalized data.
- Build a cognitive search and a health knowledge graph using AWS AI services.

Document History for the Amazon HealthLake Developer Guide

The following table describes the documentation for this release of Amazon HealthLake.

- **API version: latest**
- **Latest documentation update:** 7/14/2020

update-history-change	update-history-description	update-history-date

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.