aws

Developer Guide

AWS HealthLake

AWS HealthLake: Developer Guide

Table of Contents

What is AWS HealthLake?

> **Note**
>
> After February 20, 2023, HealthLake data stores do *not* use integrated natural language processing (NLP) by default. If you are interested in turning on this feature on your data store, see How do I turn on HealthLake's integrated natural language processing feature? in the Troubleshooting chapter.

AWS HealthLake is a HIPAA eligible service for clinical data ingestion, storage, and analysis utilizing the Healthcare Interoperability FHIR (R4) specification.

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 way that is secure, compliant, and can be audited.

HealthLake is also integrated with Amazon Athena and AWS Lake Formation. You can use this integration to query your data store using SQL.

Benefits of AWS HealthLake

With AWS 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.
- **Use the FHIR REST API operations** – HealthLake supports using the FHIR REST API operations to perform CRUD (Create/Read/Update/Delete) operations on your data store. FHIR search is also supported.

- **Store your data in the AWS Cloud in a secure, HIPAA-eligibile manner that can be audited** – You can store data in the 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.
- **Athena integration** – HealthLake's integration with Athena means you can create powerful SQL-based queries that you can use to create and save complex filter criteria. Then, you can use this data in downstream applications such as SageMaker to train a machine learning model or Amazon QuickSight to create dashboards and data visualizations.
- **Transform unstructured data using specialized machine learning (ML) models** – HealthLake provides integrated medical natural language processing (NLP) using Amazon Comprehend Medical. Raw medical text data is transformed using specialized ML models. These models 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. HealthLake then creates new resources based on the traits sign, symptom, and condition. These are added as new Condition, Observation, and MedicationStatement resource types.

HealthLake use cases

You can use HealthLake for the following healthcare applications:

- **Population health management** – HealthLake helps healthcare organizations analyze population health trends, outcomes, and costs. This helps organizations 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.
- **Optimizing hospital efficiency** – HealthLake offers hospitals key analytics and machine learning tools to improve efficiency and reduce hospital waste.

Accessing HealthLake

You can access 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 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 AWS HealthLake pricing page. To better estimate potential costs associated with HealthLake, you can use the HealthLake pricing calculator.

How AWS HealthLake works

> **Note**
>
> After February 20, 2023, HealthLake data stores do *not* use integrated natural language processing (NLP) by default. If you are interested in turning on this feature on your data store, see How do I turn on HealthLake's integrated natural language processing feature? in the Troubleshooting chapter.

AWS HealthLake creates a data store that stores health records utilizing the Healthcare Interoperability FHIR (R4) specification. With HealthLake, you can perform the following tasks.

- Create, monitor, and delete a data stores.
- Use `StartFHIRImportJob` to import healthcare data in bulk from an Amazon Simple Storage Service (Amazon S3) bucket into a data store.
- Use Create, Read, Update, and Delete (CRUD) operations to manage data stored in your data store.
- Use SQL in Amazon Athena to query your data store.
- Use an HTTP client in the FHIR REST API operations to search your data store.
- Enable the Amazon Comprehend Medical API operations to search for medical insights in your data using natural language processing (NLP).

Creating and monitoring data stores

With HealthLake, you can create and monitor data stores that can store Fast Healthcare Interoperability Resources (FHIR) data.

To create a new data store, you can use the CreateFHIRDatastore or the HealthLake console. To see the status of a data store, use DescribeFHIRDatastore. To see the status of multiple active data stores, use ListFHIRDatastores. To delete a data store, use DeleteFHIRDatastore.

Using Create, Read, Update, and Delete (CRUD) operations

You can use the FHIR REST API operations to perform Create, Read, Update, Delete (CRUD) operations on your HealthLake data store. To learn more about how HealthLake supports the FHIR

REST API operations, see Managing and searching resources in AWS HealthLake by using FHIR REST API operations.

Automated Resource generation from FHIR DocumentReference resource extensions

> **Note**
>
> When you create a HealthLake data store and add data that contains the `DocumentReference`, you will incur charges in your AWS account. For more details, see AWS HealthLake pricing.

HealthLake provides NLP on documents found in the `DocumentReference` resource type. To analyze the text, HealthLake uses the following Amazon Comprehend Medical API operations.

- `DetectEntitiesV2`: Inspects the clinical text for a variety of medical entities and returns specific information about them, such as entity category, location, and confidence score.
- `InferICD10CM`: Inspects the clinical text to detect medical conditions as entities listed in a patient record and links those entities to normalized concept identifiers in the ICD-10-CM knowledge base from the Centers for Disease Control.
- `InferRxNorm`: Inspects the clinical text to detect medications as entities listed in a patient record and links to the normalized concept identifiers in the RxNorm database from the National Library of Medicine.

HealthLake automatically analyzes data found in the `DocumentReference` resource type when it is added to your data store. The original `DocumentReference` resource files stay unchanged. The extracted medical information is automatically appended as FHIR-compliant extensions. To learn more about how NLP works in HealthLake, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake.

Querying a data store by using SQL

> **Note**
>
> For data stores created before November, 14, 2022, your search is limited to the FHIR REST API operations. To use SQL-based queries for data in your HealthLake data store, see Getting started with AWS HealthLake.

Amazon Athena is a serverless SQL-based query service. HealthLake data stores are ingested into Athena as Apache Iceberg tables. These tables are designed to support large analytic datasets. In Athena, each FHIR resource type is represented as a table. Using Athena, you can only make READ requests on your data store. To learn more about SQL-based searching, see Query your HealthLake data store using SQL.

Searching a data store using FHIR REST API operations

You can search the health records stored in your data store either by specifying a resource type with supported search parameters, or by using a resource ID found in the server, without specifying the resource type. To learn more about searching by using the FHIR REST API operations, see Managing and searching resources in AWS HealthLake by using FHIR REST API operations.

Importing data

Use AWS HealthLake to import your files in bulk from an Amazon S3 bucket. Use either the console or StartFHIRImportJob to begin an import job. After importing your files, you can use DescribeFHIRImportJob to monitor the status of the job. After the import job is complete, the data can then be added to Athena, transformed, or analyzed and used in downstream applications.

Exporting data

Use HealthLake to export your files in bulk to an Amazon S3 bucket. Use either the console or StartFHIRExportJob to begin an export job. After exporting your files, you can use DescribeFHIRExportJob to monitor the status of the job and view its properties. After the export job is complete, you can visualize the data by using Amazon QuickSight or you can access it by using other AWS services.

Getting started with AWS HealthLake

In this chapter, you use the AWS Management Console to set up permissions, create a data store, import resources, and configure an IAM user or role to be a data lake administrator in AWS Lake Formation. The data lake administrator grants access Lake Formation resources needed to use Amazon Athena to query a data store.

As an alternative to using the AWS Management Console, you can perform many of the same tasks highlighted in this exercise using the AWS Command Line Interface or the AWS SDKs. Before you use the AWS Command Line Interface or SDKs, download and configure them. See AWS Command Line Interface, AWS SDK for Python, or the AWS SDK for Java for more information.

The sections in this chapter walk you through all the steps required to get started with HealthLake.

Prerequisites: Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services.

If you are a new AWS customer, you can get started with AWS HealthLake at no charge. 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.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

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

Create an IAM user

Services in AWS, such as 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, because you will need to add IAM policies to IAM users roles.

To create an administrator and sign in to the console

1. Create an IAM 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*.

A IAM user or role with `AdministratorAccess` is needed to add an IAM user or role as a data lake administator in AWS Lake Formation.

For more information about IAM, see the following:

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

Step 1: Configuring a new IAM user or role to use HealthLake (IAM Administrator)

> **Persona: IAM Administrator**
>
> A user who can create IAM users and roles, and can add data lake administrators.

These steps in this topic must be carried out by an IAM administrator.

To connect your HealthLake data store to Athena, you need to provision an IAM user or role that is a data lake administrator and is a HealthLake administrator. This new user or role grants access to resources found in a data store via AWS Lake Formation, and has the `AmazonHealthLakeFullAccess` AWS managed policy added to their user or role. Follow these instructions to prepare an IAM user or role that has access to both HealthLake and is data lake administrator in AWS Lake Formation.

> **Important**
>
> An IAM user or role that is a data lake administrator *cannot* create new data lake administrators. To add addittional data lake administrator you must use a IAM user or role which has been granted `AdministratorAccess` access.

Provision an IAM user or role to be a data lake administrator and a HealthLake administrator

1. Add the following IAM AWS managed policy to a user or role in your organization: **`AmazonHealthLakeFullAccess`**

   - If you're unfamiliar with creating an IAM user, see Creating an IAM User and Overview of AWS IAM Policies in the *IAM User Guide*.

2. Grant the IAM user access to AWS Lake Formation.

   - Add the following IAM AWS managed policy to a user or role in your organization: **`AWSLakeFormationDataAdmin`**

     > **Note**
     >
     > The `AWSLakeFormationDataAdmin` policy grants access to all AWS Lake Formation resources. We recommend that you always use the minimum permissions required to accomplish your task. For more information, see IAM Best Practices in the *IAM User Guide*.

3. Create a service role and add it to a user or role in your oranization.

- Add the following inline policy to a user or role in your organization. To learn more about adding inline policies, see Step 2: Create a service role and add it to an IAM user or role (IAM Administrator).

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "s3:PutObject",
    "s3:GetObject"
   ],
   "Resource": "arn:aws:s3:::my-bucket/*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "ram:GetResourceShareInvitations",
    "ram:AcceptResourceShareInvitation",
    "glue:CreateDatabase",
    "glue:DeleteDatabase"
    ],
   "Resource": "*"
  }
 ]
}
```

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*.

For more information on the `AWSLakeFormationDataAdmin` policy, see Lake Formation Personas and IAM Permissions Reference in the *AWS Lake Formation Developer Guide*.

Step 2: Create a service role and add it to an IAM user or role (IAM Administrator)

Persona: IAM Administrator

A user who can create IAM users and roles, and can add data lake administrators.

For HealthLake to integrate with Athena, you need the following service role. This service role allows HealthLake to manage sharing your data store with Athena via AWS Lake Formation.

To embed an inline policy for a service role (IAM Console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. In the list, choose the name of the role that you want to edit.
4. Choose the **Permissions** tab.
5. Choose **Add inline policy**.

   Note

   You cannot embed an inline policy in a service-linked role in IAM.

6. Choose the **JSON** tab.
7. Enter the following JSON policy document. For details about the IAM policy language, see IAM JSON Policy Reference in the *IAM User Guide*.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "s3:PutObject",
    "s3:GetObject"
   ],
   "Resource": "arn:aws:s3:::my-bucket/*"
```

```
  },
  {
   "Effect": "Allow",
   "Action": [
    "ram:GetResourceShareInvitations",
    "ram:AcceptResourceShareInvitation",
    "glue:CreateDatabase",
    "glue:DeleteDatabase"
    ],
   "Resource": "*"
  }
 ]
}
```

8. When you are finished, choose **Review policy**. The Policy Validator reports any syntax errors.
9. On the **Review policy** page, enter a **Name** for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.
10. After you create an inline policy, it is automatically embedded in your role.

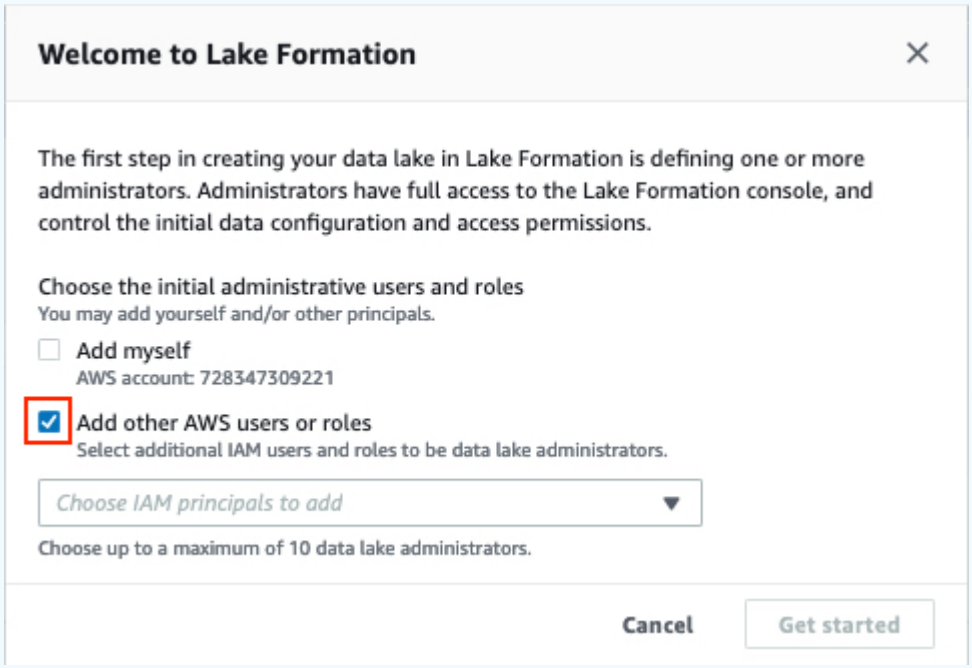
Step 3: Add a Data Lake Administrator in Lake Formation (IAM Administrator)

Next, the IAM administrator needs to add the user or role created in step 1 as a data lake administrator in Lake Formation.

To add an IAM user or role as a data lake administrator

1. Open the AWS Lake Formation console: https://console.aws.amazon.com/lakeformation/

> **Note**
>
> If this is your first time visiting Lake Formation, a **Welcome to Lake Formation** dialog box appears asking you to define a Lake Formation administrator.



2. Assign the new user or role to be a AWS Lake Formation data lake administrator.
   - *Option 1:* If you received the **Welcome to Lake Formation** dialog box.
     1. Choose **Add other AWS users or roles**.
     2. Choose the **down arrow** (▾).
     3. Choose the HealthLake administrator you would like to also be Lake Formation administrators.
     4. Choose **Get started**.
   - *Option 2:* Use the **Navigation pane** (☰).
     1. Choose the **Navigation pane** (☰).
     2. Under **Permissions**, choose **Administrative roles and tasks**.
     3. In the **Data lake administrators** section, select **Choose administrators** .
     4. In the **Manage data lake administrators** dialog box, choose the **down arrow** (▾).
     5. Next, select or search for the HealthLake administrators users or roles who you also want to be Lake Formation administrators.
     6. Then, choose **Save**.
3. Change the default security settings to be managed by Lake Formation. The HealthLake data store resources need to be managed by Lake Formation *not* IAM. To update, see Change the default permission model in the *AWS Lake Formation Developer Guide*.

Step 4: Create a data store (HealthLake Administrator)

Persona: HealthLake Administrator

A user who can create IAM users and roles. Has the `AdministratorAccess` AWS managed policy. Has all permissions on all Lake Formation resources. Can add data lake administrators. Cannot grant Lake Formation permissions if not also designated a data lake administrator.

This exercise creates a data store and pre-populates it using Synthea data. It uses the IAM user or role you created in step 1. Synthea is preloaded sample data made available by AWS HealthLake.

To create HealthLake data store (AWS Management Console)

1. Open the HealthLake console at https://console.aws.amazon.com//healthlake/home.
2. Open the **Navigation pane** (≡).
3. Then, choose **Data Stores**.
4. Next, choose **Create Data Store**.
5. In the **Data Store settings** section, for **Data Store name** specify a name.
6. (Optional) In the **Data Store settings** section, for **Preload sample data** select the checkbox to preload Synthea data.
   - Synthea data is a preloaded sample dataset. For more information, see Preloaded data types.
7. In the **Data Store encryption** section choose either **Use AWS owned key (default)** or **Choose a different AWS KMS key (advanced)**.

   Note

   We recommend that customers use a customer managed key for data stores that contain Personally identifiable information.
8. In the **Tags - *optional*** section, you can add tags to your data store.
   - To learn more about tagging your data store, see Adding a tag to a data store.
9. Next, choose **Create Data Store**.

When your data store is ready the status changes to **Ready**.

Step 5: Perform a search using SQL in Amazon Athena (HealthLake Administrator)

After you create a data store, and you populate it with preloaded data or import data, you can start querying your data store using SQL in Amazon Athena. To access your data in Athena, you will need to connect your data store. For more information, see Connecting your data store to Amazon Athena.

Preloaded data types

Persona: HealthLake Administrator

A user who can create IAM users and roles. Has the `AdministratorAccess` AWS managed policy. Has all permissions on all Lake Formation resources. Can add data lake administrators. Cannot grant Lake Formation permissions if not also designated a data lake administrator.

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 in 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	

AWS HealthLake supported FHIR profile validations

HealthLake supports the base FHIR R4 specification. Included in the R4 specification are FHIR Profiles. Profiles are used on a FHIR resource type to define a more specific resource type definition using constraints and/or extensions on the base resource type. For example, a FHIR Profile can identify mandatory fields such as extensions and value sets. A resource can support multiple profiles. All HealthLake data stores support using FHIR Profiles.

Adding a FHIR profile is *not* required when adding data to a HealthLake data store. If no FHIR profile is specified when a resource is added or updated then the resource is only validated against the base FHIR R4 schema.

FHIR Profiles to which a resource conforms to, are included in the resource, before it is ingested into HealthLake. HealthLake validates the specified FHIR Profiles when it is added to your HealthLake data store.

FHIR Profiles are specified in an implementation guide. HealthLake validates the FHIR Profiles defined in the following implementation guides.

Supported FHIR profiles by HealthLake

Name	Versi	Implementation guide	Capability
US Core	3.1.1	http://hl7.org/fhir/us/core/STU3.1.1/	Default
US Core	4.0.0	https://hl7.org/fhir/us/core/STU4/index.html	Supported
CARIN Blue Button	1.1.0	http://hl7.org/fhir/us/carin-bb/STU1.1/	Default
CARIN Blue Button	1.0.0	https://hl7.org/fhir/us/carin-bb/STU1/	Supported
Da Vinci Payer Data Exchange	1.0.0	https://hl7.org/fhir/us/davinci-pdex/	Default

Name	Versi	Implementation guide	Capability
Da Vinci Health Record Exchange (HRex)	0.2.0	https://hl7.org/fhir/us/davinci-hrex/2020Sep/	Default
DaVinci PDEX Plan Net	1.1.0	https://hl7.org/fhir/us/davinci-pdex-plan-net/STU1.1/	Default
DaVinci PDEX Plan Net	1.0.0	https://hl7.org/fhir/us/davinci-pdex-plan-net/STU1/	Supported
DaVinci Payer Data Exchange (PDex) US Drug Formulary	1.1.0	https://hl7.org/fhir/us/davinci-drug-formulary/STU1.1/	Default
DaVinci Payer Data Exchange (PDex) US Drug Formulary	1.0.1	https://hl7.org/fhir/us/davinci-drug-formulary/STU1.0.1/	Supported
National Health Authority's Ayushman Bharat Digital Mission (ABDM)	2.0	https://www.nrces.in/ndhm/fhir/r4/index.html	Default

Validating FHIR profiles specified in a resource

For a FHIR Profile to be validated add it to the `profile` element of individual resources using the profile URL designated in the implementation guide.

FHIR Profiles are validated when you add a new resource to your data store. To add a new resource, you can use the StartFHIRImportJob API operation, make a POST request to add a new resource, or make PUT to update an existing resource.

Example – To see which FHIR profile is referenced in a resource

The profile URL is added to the `profile` element in the `"meta" : "profile"` key-value pair. This resource was truncated for clarity.

```
            {
             "resourceType": "Patient",
             "id": "abcd1234efgh5678hijk9012",
             "meta": {
              "lastUpdated": "2023-05-30T00:48:07.8443764-07:00",
              "profile": [
               "http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient"
              ]
             }
            }
```

Example – How to reference a non-default supported FHIR profile

To validate against a supported non-default profile (eg. CarinBB 1.0.0) - add the profile URL with version (separated by '|') and the base profile URL in the `meta.profile` element. This example resource was truncated for clarity.

```
            {
             "resourceType": "ExplanationOfBenefit",
             "id": "sample-EOB",
             "meta": {
              "lastUpdated": "2024-02-02T05:56:09.4+00:00",
              "profile": [
               "http://hl7.org/fhir/us/carin-bb/StructureDefinition/C4BB-
ExplanationOfBenefit-Pharmacy|1.0.0",
                "http://hl7.org/fhir/us/carin-bb/StructureDefinition/C4BB-
ExplanationOfBenefit-Pharmacy“
              ]
             }
            }
```

Using an AWS HealthLake data store with Fast Healthcare Interoperability Resources (FHIR) data

In AWS HealthLake, you use a data store to store data in HL7 FHIR (R4) format. The topics in this chapter describe how to create a data store, import data into a data store, export data from a data store, and how to monitor a data store. To perform many of the actions and operations described in this chapter, you must have the required IAM permissions added to your IAM user or role. To learn more about how HealthLake interacts with IAM, see How AWS HealthLake works with IAM.

HealthLake is also integrated with AWS CloudTrail. You can use CloudTrail to provide a record of actions taken by a user, role, or an AWS service in HealthLake. CloudTrail captures all API calls and console actions for HealthLake as events. To learn more, see Logging AWS HealthLake API Calls with AWS CloudTrail.

To learn more about the Fast Healthcare Interoperability Resources (FHIR) resource types that are supported by HealthLake, see Supported FHIR resource types in AWS HealthLake.

> ⚠ **Amazon Athena compatibility**
>
> HealthLake date stores created prior to November, 14, 2022 *cannot* perform SQL queries using Athena. To use Athena search capabilities on your preexisting data store, first migrate the data to a new data store. To learn more about migrating preexisting data stores, see Migrating an existing data store to use Amazon Athena.

How to use your HealthLake data store

- Creating a HealthLake data store
- Learn more about a specific HealthLake data store
- Using the ListFHIRDatastores API operation
- Deleting a data store example
- Importing files into HealthLake data stores
- Exporting files from a HealthLake data store

Creating a HealthLake data store

After November, 14, 2022, the IAM requirements to access HealthLake changed. To both create analytics enabled data stores and to grant access to them in Athena, add the `AWSLakeFormationDataAdmin` managed policy to your IAM user, group or role. The `AWSLakeFormationDataAdmin` policy allows you to create data lake administrators and to grant access to data stores in Athena.

The status of a data store is available on the **Data stores** page in the console. A HealthLake data store can have the following statuses:

- **Creating** – Your data store is being created.
- **Active** – Your data store is active. You can import and export data from it. You can also manage and search the FHIR resources you have stored in the data store.
- **Deleting** – Your data store is being deleted.
- **Deleted** – Your data store has been deleted.

> ⚠ **HealthLake console differences**
>
> The HealthLake console does not support creating a SMART on FHIR enabled data store. To create a SMART on FHIR enabled data store, you must use the AWS CLI or one of the AWS supported SDKS. To learn more, see Integrating SMART on FHIR with AWS HealthLake. Also, the console does *not* differentiate between the two types of data stores supported by HealthLake when you view an individual data store's details page.

To create a HealthLake data store (AWS Management Console)

1. Open the HealthLake console at https://console.aws.amazon.com//healthlake/home.
2. Open the **Navigation pane** (≡).
3. Then, choose **Data Stores**.
4. Next, choose **Create Data Store**.
5. In the **Data Store settings** section, for **Data Store name** specify a name.
6. (Optional) In the **Data Store settings** section, for **Preload sample data** select the check box to preload Synthea data.

- Synthea data is a preloaded sample dataset. For more information, see Preloaded data types.

7. In the **Data Store encryption** section, choose either **Use AWS owned key (default)** or **Choose a different AWS KMS key (advanced)**.
8. In the **Tags - *optional*** section, you can add tags to your data store.
   - To learn more about tagging your data store, see Adding a tag to a data store.
9. Next, choose **Create Data Store**.

The status of your data stores are available on the **Data stores** page. A HealthLake data store can have the following statuses:

- **Creating** – Your data store is being created.
- **Active** – Your data store is active. You can import and export data from it. You can also manage and search the FHIR resources in the data store.
- **Deleting** – Your data is being deleted.
- **Deleted** – Your data store has been deleted. This cannot be undone.

To create a HealthLake data store (AWS CLI and SDKs)

You can use the following code examples to create a HealthLake data store.

AWS CLI

The following example demonstrates using the `CreateFHIRDatastore` operation with the AWS CLI. To run the example, you must install the AWS CLI. When you create your data store, encryption at rest defaults to an AWS-owned KMS key, unless specified otherwise. To learn more about encryption at REST for HealthLake see, Encryption at REST for AWS HealthLake.

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 "your-data-store-name"
```

When successful, you get the following JSON response. When your data store is ready to ingest data, the status changes to `ACTIVE`. To learn more about importing data to your HealthLake data store, see Importing files into HealthLake data stores.

```
{
    "DatastoreId": "eeb8005725ae22b35b4edbdc68cf2dfd",
 "DatastoreArn": "arn:aws:healthlake:us-west-2:111122223333:datastore/fhir/
eeb8005725ae22b35b4edbdc68cf2dfd",
    "DatastoreStatus": "CREATING",
    "DatastoreEndpoint": "https://healthlake.us-west-2.amazonaws.com/datastore/
eeb8005725ae22b35b4edbdc68cf2dfd/r4/"
}
```

To view a list of all data storesdata stores, you can use the `ListFHIRDataStore` operation. You can also see a list of **Active** data stores in the HealthLake console.

Python (boto3)

The following example demonstrates how to create a HealthLake data store using the `create_fhir_datastore` operation. When you create your data store encryption at rest defaults to an AWS-owned AWS KMS key unless specified otherwise. To learn more about encryption at REST for HealthLake see, Encryption at REST for AWS HealthLake.

```
import boto3
import logging #built in logging library
from botocore.exceptions import ClientError, ValidationError #specific exception
 ClientError from the boto3 library

def create_healthlake_datastore(DatastoreName=None):
    '''
    :param DatastoreName: the name of the data store, string
    :param:
    :return: True if the data store is created, else False
    '''

    # Create an Amazon Healthlake data store
    # Should we say something about region setting?
    # Should this example have some handling KMS keys

    try:
        if DatastoreName is None:
            healthlake_client = boto3.client('healthlake')
            healthlake_client.create_fhir_datastore(DatastoreTypeVersion='R4')
```

```
        else:
            healthlake_client = boto3.client('healthlake')
            healthlake_client.create_fhir_datastore(DatastoreTypeVersion='R4',
                                                    DatastoreName=DatastoreName)
    except (ClientError, ValidationError) as e:
        logging.error(e)
        return False

    return True

# Run the function above
create_healthlake_datastore(DatastoreName='test-datastore-delete-me-2')
```

A data store can have one of four statuses. Use `list_fhir_datastores` to view a list of your HealthLake data stores regardless of status. This example shows how you can filter based on the status of a data store.

```
import boto3

healthlake_client = boto3.client('healthlake')
data_store_list = healthlake_client.list_fhir_datastores(Filter={'DatastoreStatus':
 'ACTIVE'})
print(data_store_list)
```

To learn more, see `list_fhir_datastore` in the *Boto3 Documentation*.

Java

The following example demonstrates how to create HealthLake data store using the `CreateFHIRDatastoreRequest` operation. To run the example, you must install the AWS SDK for Java. When you create your data store encryption at rest defaults to an AWS-owned AWS KMS key unless specified otherwise. To learn more about encryption at REST for HealthLake see, Encryption at REST for AWS HealthLake.

```
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));



    }
}
```

Learn more about a specific HealthLake data store

To describe an individual data store you can use AWS Management Console, AWS SDKs, and the AWS CLI.

Differences between the AWS Management Console and HealthLake APIs.

In the HealthLake console, you can only view data stores which have the status **Active**, **Creating**, or **Deleting**. To view details about a HealthLake data store which has been deleted your must-use the `DescribeFHIRDatastore` action.

To view the details of a HealthLake data store (AWS Management Console)

1. Open the HealthLake console at https://console.aws.amazon.com//healthlake/home.
2. Open the **Navigation pane** (≡).
3. Then, choose **Data Stores**.
4. On the **data stores** page, choose the name of data store you would like to learn more about.

To view the details of more than one data store at a time use the `ListFHIRDatastore` API action.

To create HealthLake data store (AWS CLI and SDKs)

You can use the code samples below to create a HealthLake data store.

AWS CLI

The following examples demonstrates using the `DescribeFHIRDatastore` operation wit the AWS CLI. To run the example, you must install the AWS CLI.

```
aws healthlake describe-fhir-datastore --datastore-id
 "5b6e4cd798289a4ab8dad6c1002dd731"
```

When successful, you get the following JSON response.

```
{
    "DatastoreProperties": {
     "DatastoreId": "eeb8005725ae22b35b4edbdc68cf2dfd",
        "DatastoreArn": "arn:aws:healthlake:us-west-2:728347309221:datastore/
fhir/5b6e4cd798289a4ab8dad6c1002dd731",
        "DatastoreName": "delete-me",
        "DatastoreStatus": "ACTIVE",
        "CreatedAt": "2022-10-03T10:53:45.020000-07:00",
```

```
        "DatastoreTypeVersion": "R4",
        "DatastoreEndpoint": "https://healthlake.us-west-2.amazonaws.com/
datastore/5b6e4cd798289a4ab8dad6c1002dd731/r4/",
        "SseConfiguration": {
            "KmsEncryptionConfig": {
                "CmkType": "AWS_OWNED_KMS_KEY"
            }
        },
        "PreloadDataConfig": {
            "PreloadDataType": "SYNTHEA"
        }
    }
}
```

Python (boto3)

The AWS SDK for Python supports the `describe_fhir_datastore` method which takes in a single parameter `DatastoreId`.

```
import boto3

#Create a Healthlake client
healthlake_client = boto3.client('healthlake')

#Call the describe_fhir_datastore method
data_store_details =
 healthlake_client.describe_fhir_datastore(DatastoreId='cdf8f1557e57c543bdc627fb8f12b7fd')

print(data_store_details)
```

When successful, it returns a python dictionary.

```
{'DatastoreProperties': {'DatastoreId': 'cdf8f1557e57c543bdc627fb8f12b7fd',
 'DatastoreArn': 'arn:aws:healthlake:us-west-2:728347309221:datastore/fhir/
cdf8f1557e57c543bdc627fb8f12b7fd', 'DatastoreName': '08-24-2022-test-data-
store', 'DatastoreStatus': 'ACTIVE', 'CreatedAt': datetime.datetime(2022,
 8, 23, 22, 12, 14, 359000, tzinfo=tzlocal()), 'DatastoreTypeVersion': 'R4',
 'DatastoreEndpoint': 'https://healthlake.us-west-2.amazonaws.com/datastore/
cdf8f1557e57c543bdc627fb8f12b7fd/r4/', 'SseConfiguration': {'KmsEncryptionConfig':
 {'CmkType': 'AWS_OWNED_KMS_KEY'}}, 'PreloadDataConfig': {'PreloadDataType':
 'SYNTHEA'}}, 'ResponseMetadata': {'RequestId': 'aef4b268-ad4b-4b57-
bc97-2da956356835', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Wed, 05 Oct
 2022 01:21:44 GMT', 'content-type': 'application/x-amz-json-1.0', 'content-
```

```
length': '547', 'connection': 'keep-alive', 'x-amzn-requestid': 'aef4b268-ad4b-4b57-
bc97-2da956356835'}, 'RetryAttempts': 0}}
```

To return details about more than one data store at a time use `ListFHIRDatastore`

To view details about more than one HealthLake data store at a time use the ListFHIRDatastores API operation.

Using the ListFHIRDatastores API operation

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 HealthLake data store you can use the AWS Management Console, AWS Management Console, AWS SDKs, and the AWS CLI.

Deleting a data store is an asynchronous operation. Once started, the status changes to **Deleting**. A data store maintains the status of **Deleting** until all the FHIR data from the date store, and underlying infrastructure necessary are removed as well.

Once the data and infrastructure are removed, your HealthLake data store status changes to **Deleted**. After deletion, the details about your data stores are available only by using the `DescribeFHIRDataStore` and `ListFHIRDataStores` operations for seven days. After seven days, the deleted data store will not appear in the results.

To successfully delete a data store the user, group, or role making the request must have the IAM action `glue:DeleteDatabase` added to their IAM policy. This IAM action is not included as part of the AWS managed policy, `AmazonHealthLakeFullAccess`.

To delete a HealthLake data store (AWS Management Console)

1. Open the HealthLake console at https://console.aws.amazon.com//healthlake/home.
2. Open the **Navigation pane** (≡).
3. Then, choose **Data Stores**.
4. On the **Data Stores** page, choose the option next to the data store you want to delete.
5. Then, choose **Delete**
6. In the dialog box type **`delete`** to confirm that you want to delete the select data store.
7. Then, choose **Delete**.

Then the status of your data store will change from **Active** to **Deleting**.

To delete a HealthLake data store (AWS CLI and SDKs)

You can use the code samples below to delete a HealthLake data store.

AWS CLI

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

```
aws healthlake delete-fhir-datastore --datastore-id
 'eeb8005725ae22b35b4edbdc68cf2dfd'
```

When successful, you get the following JSON response.

```
{
    "DatastoreProperties": {
     "DatastoreId": "eeb8005725ae22b35b4edbdc68cf2dfd",
        "DatastoreArn": "arn:aws:healthlake:us-west-2:728347309221:datastore/fhir/",
        "DatastoreName": "delete-me",
        "DatastoreStatus": "ACTIVE",
        "CreatedAt": "2022-10-03T10:53:45.020000-07:00",
        "DatastoreTypeVersion": "R4",
        "DatastoreEndpoint": "https://healthlake.us-west-2.amazonaws.com/
datastore/5b6e4cd798289a4ab8dad6c1002dd731/r4/",
        "SseConfiguration": {
            "KmsEncryptionConfig": {
                "CmkType": "AWS_OWNED_KMS_KEY"
            }
        },
        "PreloadDataConfig": {
            "PreloadDataType": "SYNTHEA"
        }
    }
}
```

Python (boto3)

The AWS SDK for Python supports the `describe_fhir_datastore` method which takes in a single parameter `DatastoreId`.

```
import boto3
```

```
#Create a Healthlake client
healthlake_client = boto3.client('healthlake')

#Call the describe_fhir_datastore method
data_store_details =
 healthlake_client.describe_fhir_datastore(DatastoreId='cdf8f1557e57c543bdc627fb8f12b7fd')

print(data_store_details)
```

When successful, it returns a python dictionary.

```
{'DatastoreProperties': {'DatastoreId': 'cdf8f1557e57c543bdc627fb8f12b7fd',
 'DatastoreArn': 'arn:aws:healthlake:us-west-2:728347309221:datastore/fhir/
cdf8f1557e57c543bdc627fb8f12b7fd', 'DatastoreName': '08-24-2022-test-data-
store', 'DatastoreStatus': 'ACTIVE', 'CreatedAt': datetime.datetime(2022,
 8, 23, 22, 12, 14, 359000, tzinfo=tzlocal()), 'DatastoreTypeVersion': 'R4',
 'DatastoreEndpoint': 'https://healthlake.us-west-2.amazonaws.com/datastore/
cdf8f1557e57c543bdc627fb8f12b7fd/r4/', 'SseConfiguration': {'KmsEncryptionConfig':
 {'CmkType': 'AWS_OWNED_KMS_KEY'}}, 'PreloadDataConfig': {'PreloadDataType':
 'SYNTHEA'}}, 'ResponseMetadata': {'RequestId': 'aef4b268-ad4b-4b57-
bc97-2da956356835', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Wed, 05 Oct
 2022 01:21:44 GMT', 'content-type': 'application/x-amz-json-1.0', 'content-
length': '547', 'connection': 'keep-alive', 'x-amzn-requestid': 'aef4b268-ad4b-4b57-
bc97-2da956356835'}, 'RetryAttempts': 0}}
```

To return details about more than one data store at a time use `ListFHIRDatastore`

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.

```
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/eeb8005725ae22b35b4edbdc68cf2dfd/r4/",
    "DatastoreArn": "arn:aws:healthlake:us-east-1:(AWS Account ID):datastore/(Datastore
 ID)",
    "DatastoreStatus": "DELETING",
    "DatastoreId": "(Datastore ID)"
}
```

Importing files into HealthLake data stores

After you create your HealthLake data store, you can import files from an Amazon Simple Storage Service (Amazon S3) bucket. You can use the HealthLake console or the `StartFHIRImportJob`to start an import job. HealthLake accepts input files in newline delimited JSON (`.ndjson`) format, where each line consists of a valid FHIR resource. You can use the API operations `DescribeFHIRImportJob` and `ListFHIRImportJobs` to 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 concurrently per HealthLake data store. However, users can create, read, update, or delete FHIR resources while an import job is in progress.

For each import job, a `manifest.json` file is generated. This file describes both the successes and failures of an import job. Users can programmatically navigate to these files. They are organized into two folders, named SUCCESS and FAILURE. An output file may contain sensitive information, therefore, users must provide both an output Amazon S3 bucket and an AWS KMS key for encryption.

The following is an example of the output `manifest.json` file. It is recommended users use this 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 by using either the AWS HealthLake console or the AWS HealthLake import API, start-fhir-import-job API.

Importing files by using the API operations

Prerequisites

When you use the AWS HealthLake API operations, you must first create an AWS Identity and Access 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), use the following steps.

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, indicate to HealthLake the name of the Amazon S3 bucket that contains the input files, the KMS key you want 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 by using the console

To import files (console), use the following steps.

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. To see the status of your import job, use `ListFHIRImportJobs`. For more details on the `ListFHIRImportJobs` API command, see ListFHIRImportJobs in the *AWS HealthLake API Reference*.

IAM policies for import jobs

The IAM role that calls the AWS HealthLake API operations 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"
         "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "(accountId)"
                },
            "ArnEquals": {
                "aws:SourceArn": "arn:aws:healthlake:(region):(accountId):datastore/
fhir/(datastoreId)"
                }
            }
        }
    ]
}
```

Example: Starting and monitoring import jobs by 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.

```
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.

```
    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 HealthLake data store

To export data from your HealthLake data store, use the following operations.

- Make an export request using the `StartFHIRExportJob` API operation using the HealthLake SDK.
  - This operation only supports making a system-wide export request.
- Make an export request using the `export` syntax using the HealthLake FHIR REST API.
  - This operation supports making system-wide, Patient, and Group export requests. You can also apply parameters to further filter the data in the export request.

Both of these operations require a service role. In it, HealthLake must be defined as the service principal, and you must define an Amazon Simple Storage Service (S3) bucket of where you want to export your files. To learn more, see Creating a service role.

Both of these operations only support exporting your files to an Amazon S3 (S3) bucket. All files from your HealthLake data store are exported as newline delimited JSON (`.ndjson`) files, where each line consists of a valid FHIR resource.

To export files from your HealthLake data store, see the following sections.

- Exporting data from your data store by using the HealthLake SDK
- Exporting data from your data store by using the FHIR REST API

Exporting data from your data store by using the HealthLake SDK

You can export files from your data store to an Amazon Simple Storage Service (Amazon S3) bucket. Files from your data store are exported in newline delimited JSON (`.ndjson`) format, where each line consists of a valid FHIR resource. A 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*.

You can run one concurrent SDK-based export job for each data store per AWS account. To learn more about the Service Quotas associated with HealthLake, see AWS HealthLake endpoints and quotas.

You can still create, read, update, and delete FHIR resources while an export job is in progress.

Performing an export

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

Exporting from your data store

Prerequisites

To use AWS HealthLake API operations, 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, use the following steps.

1. Create an S3 bucket. The Amazon S3 bucket must be in the same AWS Region as the service, and Block Public Access 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"],
    "Effect": "Allow",
    "Resource": "*"
    }, {
    "Action": "iam:PassRole"
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": "healthlake.amazonaws.com"
      }
    }
  }]
}
```

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",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "your-account-id"
            },
            "ArnEquals": {
                "aws:SourceArn": "https://healthlake.your-region.amazonaws.com/
datastore/your-datastore-id/r4/"
            }
        }
    }]
}
```

5. Add a permissions 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), use the following steps.

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. Also use a KMS key encryption. To learn more about using KMS keys, see Amazon Key Management Service.
3. To see the status of your export job, use `ListFHIRExportJobs`. For more details on the `ListFHIRExportJobs` API command, see ListFHIRExportJobs in the *Amazon HealthLake API Reference*.

Exporting data from your data store by using the FHIR REST API

> **Important**
>
> HealthLake data stores created prior to June 1, 2023 only support FHIR REST API based export job requests for system-wide exports.
> HealthLake data stores created prior to June 1, 2023 do not support getting the status of an export using a `GET` request on a data store's endpoint.

To make an export request using the FHIR REST API, you must have a IAM user, group, or role with the required permissions, specify `$export` as part of the POST request, and include request parameters in the body of your request. According to the FHIR specification, the FHIR server must support GET requests, and can support POST requests. In order to support additional parameters, a body is needed to start the export, therefore HealthLake supports POST requests.

All export requests you make using the FHIR REST API are returned in `ndjson` format and exported to an Amazon S3 bucket. Each S3 object will contain only a single FHIR resource type.

You can make a single export request for each AWS account at a time. To learn more about the Service Quotas associated with HealthLake, see AWS HealthLake endpoints and quotas.

HealthLake supports the following three types of bulk export endpoint requests.

Type	Descriptions	Syntax
System export	Export all data from the HealthLake FHIR server.	`POST https://healthlake.` **`your-region`** `.amazonaws.com/datastore/` **`your-datastore-id`** `/r4/$export`
All patients	Export all data relating to all patients including resource types associated with the Patient resource type.	`POST https://healthlake.` **`your-region`** `.amazonaws.com/datastore/` **`your-datastore-id`** `/r4/Patient/$export`
Group of Patients	Export all data relating to a group of patients specified with a Group ID.	`POST https://healthlake.` **`your-region`** `.amazonaws.com/datastore/` **`your-datastore-id`** `/r4/Group/` **`ID`**`/$export`

Before you begin

Meet the following requirements to make an export request by using the FHIR REST API for HealthLake.

- You must have set up a user, group, or role that has the necessary permissions to make the export request. To learn more, see Authorizing an `export` request.
- You must have created a service role that grants HealthLake access to the Amazon S3 bucket to which you want your data to be exported. The service role must also specify HealthLake as the service principal. To learn more, see Creating a service role.

Authorizing an `export` request

To make a successful export request using the FHIR REST API, authorize your user, group, or role m by using either IAM or OAuth2.0. You also must have a service role.

Authorizing a request by using IAM

When you make an `$export` request, the user, group, or role must have `StartFHIRExportJobWithPost`, `DescribeFHIRExportJobWithGet`, and `CancelFHIRExportJobWithDelete` IAM actions included in the policy.

Authorizing a request using SMART on FHIR (OAuth 2.0)

When you make an `$export` request on SMART on FHIR enabled HealthLake data store, you need to have the appropriate scopes assigned. To learn more about supported scopes, see HealthLake data store FHIR resource specific scopes.

Making an `export` request

This section describes the required steps you must take when making an export request by using the FHIR REST API.

To avoid accidental charges on your AWS account, we recommend testing your requests by making a POST request without supplying the `export` syntax.

To make the request, you must do the following:

1. Specify `export` in the POST request URL for a supported endpoint.
2. Specify the required header parameters.
3. Specify a request body that defines the required parameters.

Step 1: Specify `export` in the POST request URL for a supported endpoint

HealthLake supports three types of bulk export endpoint requests. To make a bulk export request, you must make a POST-based request on one of the three supported endpoints. The following examples demonstrate how to specify `export` in the request URL.

- `POST https://healthlake.`**`your-region`**`.amazonaws.com/datastore/`**`your-datastore-id`**`/r4/$export`
- `POST https://healthlake.`**`your-region`**`.amazonaws.com/datastore/`**`your-datastore-id`**`/r4/Patient/$export`
- `POST https://healthlake.`**`your-region`**`.amazonaws.com/datastore/`**`your-datastore-id`**`/r4/Group/`**`ID`**`/$export`

In that POST request string, you can use the following supported search parameters.

Supported search parameters

HealthLake supports the following search modifiers in bulk export requests.

These examples include special characters which must be encoded prior to submitting your request.

Name	Required?	Description	Example
`_outputFormat`	No	The format for the requested Bulk Data files to be generated . Accepted values are `application/fhir+ndjson` , `application/ndjson` , `ndjson`.	
`_type`	No	A string of comma delimited FHIR resource types that you want included in your export job. We recommend including `_type` because this can have a cost implication when all resources are exported.	`&_type=MedicationStatement,Observation`
`_since`	No	Resource types modified on or after the date time stamp. If a resource type does *not* have a last updated time they will be included in your response.	`&_since=2024-05-09T00%3A00%3A00Z`

Step 2: Specify the required header parameters

To make an export request using the FHIR REST API, you must specify the following two header parameters.

- Content-Type: `application/fhir+json`
- Prefer: `respond-async`

Next, you must specify the required elements in the request body.

Step 3: Specify a request body the defines the required parameters.

The export request also requires a body in `JSON` format. The body can include the following parameters.

To learn more, see Creating a service role

Key	Required?	Description	Value
DataAccessRoleArn	Yes	An ARN of a HealthLake service role. The service role used must specify HealthLake as the service principal.	`arn:aws:iam::` **`444455556666`** `:role/`**`your-healthlake-service-role`**
JobName	No	The name of the export request.	**`your-export-job-name`**
S3Uri	Yes	Part of an OutputDataConfig key. The S3 URI of the destination bucket where your exported data will be downloaded.	`s3://DOC-EXAMPLE-DESTINATION-BUCKET/` **`EXPORT-JOB`** `/`
KmsKeyId	Yes	Part of an OutputDataConfig key. The ARN of the AWS KMS key used to secure the Amazon S3 bucket.	`arn:aws:kms:` **`region-of-bucket:123456789012`** `:key/`**`1234abcd-12ab-34cd-56ef-1234567890ab`**

Example – Body of an export request made by using the FHIR REST API

To make an export request by using the FHIR REST API, you must specify a body, as shown in the following.

```
{
  "DataAccessRoleArn": "arn:aws:iam::444455556666:role/your-healthlake-service-role",
  "JobName": "your-export-job",
  "OutputDataConfig": {
    "S3Configuration": {
      "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/EXPORT-JOB",
      "KmsKeyId": "arn:aws:kms:region-of-
bucket:444455556666:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  }
}
```

When your request is successful, you will receive the following response.

Response Header

```
content-location: https://healthlake.your-region.amazonaws.com/datastore/your-
datastore-id/r4/export/your-export-request-job-id
```

Response Body

```
{
  "datastoreId": "your-data-store-id",
  "jobStatus": "SUBMITTED",
  "jobId": "your-export-request-job-id"
}
```

Managing your export request

After making a successful export request, you can manage that request by using `export` to describe the status of a current export request, and `export` to cancel a current export request.

When you cancel an export request by using the REST API, you will only be billed for the portion of the data that was exported up to the time you submitted the cancel request.

The following topics describe how you can get the status on or cancel a current export request.

Canceling an export request

To cancel an export request, make a DELETE request and supply the job ID in the request URL.

```
DELETE https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
export/your-export-request-job-id
```

When your request is successful, you receive the following.

```
{
  "exportJobProperties": {
    "jobId": "your-original-export-request-job-id",
    "jobStatus": "CANCEL_SUBMITTED",
    "datastoreId": "your-data-store-id"
  }
}
```

When your request is not successful, you receive the following.

```
{
  "resourceType": "OperationOutcome",
  "issue": [
    {
      "severity": "error",
      "code": "not-supported",
      "diagnostics": "Interaction not supported."
    }
  ]
}
```

Describing an export request

To get the status of an export request, make a GET request by using `export` and your **`export-request-job-id`**.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
export/your-export-request-id
```

The JSON response will contain an `ExportJobProperties` object. It may contain the following key:value pairs.

Name	Required?	Description	Value
DataAccessRoleArn	No	An ARN of a HealthLake service role. The service role used must specify HealthLake as the service principal.	`arn:aws:iam:: 444455556666 :role/your-healthlake-service-role`
SubmitTime	No	The date time an export job was submitted.	`Apr 21, 2023 5:58:02`
EndTime	No	The time an export job was completed.	`Apr 21, 2023 6:00:08 PM`
JobName	No	The name of the export request.	**your-export-job-name**
JobStatus	No		Valid values are: `SUBMITTED | IN_PROGRESS | COMPLETED_WITH_ERRORS | COMPLETED | FAILED`
S3Uri	Yes	Part of an OutputDataConfig object. The Amazon S3 URI of the destination bucket where your exported data will be downloaded.	`s3://DOC-EXAMPLE-DESTINATION-BUCKET/` **EXPORT-JOB** `/`
KmsKeyId	Yes	Part of an OutputDataConfig object. The	`arn:aws:kms:` **region-of-**

Name	Required?	Description	Value
		ARN of the AWS KMS key used to secure the Amazon S3 bucket.	**bucket:123456789012** :key/**1234abcd-12ab-34cd-56ef-1234567890ab**

Example : Body of a describe export request made using the FHIR REST API

When successful, you will get the following JSON response.

```
{
  "exportJobProperties": {
    "jobId": "your-export-request-id",
    "JobName": "your-export-job",
    "jobStatus": "SUBMITTED",
    "submitTime": "Apr 21, 2023 5:58:02 PM",
    "endTime": "Apr 21, 2023 6:00:08 PM",
    "datastoreId": "your-data-store-id",
    "outputDataConfig": {
      "s3Configuration": {
        "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/EXPORT-JOB",
        "KmsKeyId": "arn:aws:kms:region-of-
bucket:444455556666:key/1234abcd-12ab-34cd-56ef-1234567890ab""
      }
    },
    "DataAccessRoleArn": "arn:aws:iam::444455556666:role/your-healthlake-service-role",
  }
}
```

Managing and searching resources in AWS HealthLake by using FHIR REST API operations

In AWS HealthLake, you can use Fast Healthcare Interoperability Resources (FHIR) REST API operations to manage and search resources in your HealthLake data store.

You can use FHIR REST API operations to perform Create, Read, Update, and Delete (CRUD) operations on resources in a data store. You can also form complex search strings using either a GET or POST HTTP requests. HealthLake supports a subset of FHIR-supported search operations. To learn more, see HealthLake supported search parameters.

To find the FHIR-related capabilities of an active HealthLake data store, make a GET request where `metadata` is specified in the URL, as follows.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
metadata
```

When successful, you will receive a 200 HTTP response code and the capability statement for your data store. For more information on the capability statement, see Resource CapabilityStatement - Content in the *FHIR Documentation Index*.

FHIR resource types are validated according to the FHIR R4 Structure Definition. If a resource is not valid, you will receive an OperationOutcome message with details explaining the exception.

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

Supported Operations

Operation	Description	Instance-level, Type-level or Whole-system interaction
Read	Read the current state of a 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

Operation	Description	Instance-level, Type-level or Whole-system interaction
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

Contents

Supported FHIR resource types in AWS HealthLake

This table lists the resource types supported by HealthLake.

Supported FHIR resource types

Account	DetectedIssue	Invoice	Practitioner
ActivityDefinition	Device	Library	PractitionerRole
AdverseEvent	DeviceDefinition	Linkage	Procedure
AllergyIntolerance	DeviceMetric	List	Provenance
Appointment	DeviceUseStatement	Location	Questionnaire
AppointmentResponse	DeviceRequest	Measure	QuestionnaireResponse
AuditEvent-*See note*	DiagnosticReport	MeasureReport	RelatedPerson

Binary	DocumentManifest	Media	RequestGroup
BodyStructure	DocumentReference	Medication	ResearchStudy
Bundle - *See Note*	EffectEvidenceSynthesis	MedicationAdministration	ResearchSubject
CapabilityStatement	Encounter	MedicationDispense	RiskAssessment
CarePlan	Endpoint	MedicationKnowledge	RiskEvidenceSynthesis
CareTeam	EpisodeOfCare	MedicationRequest	Schedule
ChargeItem	EnrollmentRequest	MedicationStatement	ServiceRequest
ChargeItemDefinition	EnrollmentResponse	MessageHeader	Slot
Claim	ExplanationOfBenefit	MolecularSequence	Specimen
ClaimResponse	FamilyMemberHistory	NutritionOrder	StructureDefinition
Communication	Flag	Observation	StructureMap
CommunicationRequest	Goal	OperationOutcome	Substance
Composition	Group	Organization	SupplyDelivery
ConceptMap	GuidanceResponse	OrganizationAffiliation	SupplyRequest
Condition	HealthcareService	Parameters	Task
Consent	ImagingStudy	Patient	ValueSet
Contract	Immunization	PaymentNotice	VisionPrescription
Coverage	ImmunizationEvaluation	PaymentReconciliation	VerificationResult

CoverageEligibilityRequest	ImmunizationRecommendation	Person	
CoverageEligibilityResponse	InsurancePlan	PlanDefinition	

FHIR specifications and HealthLake

- You cannot make GET or POST requests with these resource types: Binary, Bundle, OperationOutcome, and Parameters.
- **AuditEvent** — An AuditEvent resource can be created or read, but it cannot be updated or deleted.
- **Bundle** — There are multiple ways HealthLake manages Bundle requests. For more details, see Managing multiple FHIR resources using Bundle.
- **VerificationResult** — This resource type is only supported for data stores created after December 09, 2023.

Performing Create, Read, Update, and Delete (CRUD) operations on HealthLake data stores

These topics describe how to perform Create, Read, Update, and Delete (CRUD) operations on your HealthLake data store using the FHIR REST API operations.

Users must use a Signature Version 4 signing process to authenticate HealthLake API requests sent through an HTTP client. To learn more, see Signature Version 4 signing process in the *AWS General Reference*.

Topics

Creating a resource with POST

You use a POST request to create a new resource in a HealthLake data store. POST requests do not require that you provide an `id` element. The HealthLake servers return a `201` Created HTTP status code when a resource has been successfully created.

> **Note**
>
> When you make a POST request on the `DocumentReference` resource type, the existing extensions are not modified. Instead, AWS HealthLake adds the new extensions with the existing ones to your data store. For more details about how HealthLake uses natural language processing (NLP) on the `DocumentReference` resource type to extract valuable medical data, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake.

Example Creating a `Patient` resource using a POST request.

To create a HealthLake data store POST request, use your data store's endpoint and provide a JSON request body. To find a data store's endpoint, look in the HealthLake console under **Data Stores** or by using the DescribeFHIRDatastore operation in the *AWS HealthLake API Reference*.

POST Request

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient
```

JSON Request Body

```
{
  "resourceType": "Patient",
  "identifier": [ { "system": "urn:oid:1.2.36.146.595.217.0.1", "value":
 "12345" } ],
  "name": [ {
      "family": "Silva",
      "given": ["Ana", "Carolina"]
  } ],
  "gender": "female",
  "birthDate": "1992-02-10"
}
```

JSON Response

To confirm the creation of the patient resource, you will receive a 201 Created HTTP status code and the following JSON response.


```
{
  "resourceType": "Patient",
  "identifier": [
    {
      "system": "urn:oid:1.2.36.146.595.217.0.1",
      "value": "12345"
    }
  ],
  "name": [
    {
      "family": "Silva",
      "given": [
        "Ana",
        "Carolina"
      ]
    }
  ],
  "gender": "female",
  "birthDate": "1992-02-10",
  "id": "274b408a-1201-4e9f-a621-1df937f1a26d",
  "meta": {
    "lastUpdated": "2022-06-13T23:31:24.427Z"
  }
}
```


Reading a resource with GET

This example shows you how to read a patient FHIR resource using a GET request.

Example Reading a specific `Patient` resource using a `GET` request.

To create a HealthLake data store GET request, use your data store's endpoint. To find a data store's endpoint, look in the HealthLake console under **Data Stores** or by using the DescribeFHIRDatastore operation in the *AWS HealthLake API Reference*.

You also must include the resource type, **`Patient`** and a valid identifier, **`2de04858-ba65-44c1-8af1-f2fe69a977d9`**.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9
```

JSON Response

When successful, you will receive a 200 HTTP status code and 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",
    "meta": {
        "lastUpdated": "2020-11-23T06:24:13.202Z"
    },
    "id": "2de04858-ba65-44c1-8af1-f2fe69a977d9"
}
```

Updating a resource using PUT

The following example shows you how to use PUT to update details about a patient in the patient FHIR resource type. Furthermore, when you make a PUT request on a resource not yet created, it will create an initial version.

You request will return either a 200 HTTP status code if the resource was updated, or it will return a 201 HTTP status code if a new resource was created.

> **Note**
>
> When you make a PUT request on the `DocumentReference` resource type, the existing extensions are not modified. Instead, AWS HealthLake adds the new extensions with the existing ones to your data store. For more details about how HealthLake uses natural language processing (NLP) on the `DocumentReference` resource type to extract valuable medical data, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake.

Example Updating a `Patient` resource type using a PUT request

When you make a PUT request, you will need the data store's endpoint, the name of the resource type you want to update, an identifier, and a JSON request body.

If you use PUT to create a new resource, it uses the identifier provided to create the new resource.

PUT Request

Example structure of a valid PUT request:

```
PUT https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9
```

JSON Request Body

An example JSON body used to update the specified patient resource.

```
{
  "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"
}
```

JSON response

You will 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"
 }
}
```

Conditional Update

Conditional Update allows updating an existing resource based on some identification search criteria, rather than by logical id. When the server processes this update, it performs a search using

its standard search capabilities for the resource type, with the goal of resolving a single logical id for this request.

The action it takes depends on how many matches are found:

- **No matches, no id provided in the request body**: The server creates the resource.
- **No matches, id provided and resource doesn't already exist with the id**: The server treats the interaction as an Update as Create interaction.
- **No matches, id provided and already exist**: The server rejects the update with a `409 Conflict` error.
- **One Match, no resource id provided OR (resource id provided and it matches the found resource)**: The server performs the update against the matching resource as above where, if the resource was updated, the server SHALL return a `200 OK`;
- **One Match, resource id provided but does not match resource found**: The server returns a `409 Conflict` error indicating the client id specification was a problem preferably with an OperationOutcome
- **Multiple matches**: The server returns a `412 Precondition Failed` error indicating the client's criteria were not selective enough preferably with an OperationOutcome

Example – Update a patient resource whose name is peter, birthdate is 1st Jan 2000 and phone number 1234567890:

```
PUT https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?name=peter&birthdate=2000-01-01&phone=1234567890
```

Deleting a resource using DELETE

To delete a resource in your HealthLake data store, you must make a `DELETE` HTTP request.

Example Deleting a specific `Patient` resource type using a `DELETE` request.

To create a `DELETE` request, use the data store's endpoint. To find a data store's endpoint, look in the HealthLake console under **Data Stores** or by using the DescribeFHIRDatastore operation found in the *AWS HealthLake API Reference*.

You also must include the resource type and a valid identifier.

```
DELETE https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/2de04858-ba65-44c1-8af1-f2fe69a977d9
```

HTTP response

When successful, you will receive a 204 HTTP status code confirming that the resource is no longer in the data store. When a delete request fails you will receive a 400 series HTTP status code indicating why the DELETE request failed.

Managing multiple FHIR resources using Bundle

In the HL7 FHIR R4 specification, bundles are simply a collection of resources. HealthLake supports creating a Bundle resource type in a FHIR REST API request, and using a bundle transaction to perform multiple CRUD operations in a single FHIR REST API request. In a bundle transaction, you must specify the bundle type as `batch` in the FHIR REST API request.

All bundle requests are recorded by AWS CloudTrail. To learn more about using CloudTrail with HealthLake, see Logging AWS HealthLake API Calls with AWS CloudTrail.

HL7 FHIR R4 Resources (External)

- To read the full specification, see Resource Type: Bundle in the *FHIR Documentation Index*.
- To read about batch interactions using the FHIR REST API, see Batch interactions using the FHIR REST API in the *FHIR Documentation Index*.

The sections below describe how to structure a FHIR REST API request in order to either create a new Bundle resource or to process resources individually using bundle transactions.

> ⚠ **Differences between the HealthLake console, the AWS CLI, and the AWS SDKs**
>
> The HealthLake console only supports Bundle type operations where the Bundle resource type is specified in the FHIR REST API request URL.

Performing multiple CRUD operations using FHIR bundles

When *no* resource type is specified in your request URL, the FHIR REST API request is parsed as individual data store transactions. Each CRUD operation provided in the JSON body is evaluated and a specific HTTP status code returned. HealthLake supports the Bundle type `batch`.

To perform multiple CRUD operations in a single FHIR REST API request do the following:

The following list shows truncated portions of a request body used in bundle FHIR REST API request. For a full request body, see Creating a bundle request involving multiple CRUD operations.

1. Do *not* specify a resource type in your POST request:

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
```

2. In the request body specify the Bundle type as `"type": "batch"`
3. In the request body specify resource-specific data for each CRUD interaction starting with the `resource` key.
4. Each CRUD operation is specified as a `request` in the request body as follows:

```
{ ...
  "request" : {
    "method" : "HTTP-VERB",
    "url" : "FHIR-RESOURCE-TYPE-URL"
  }
  ...
}
```

In the JSON response, you get an HTTP status code for each CRUD operation specified in the request.

HealthLake limits Bundle transactions

- To learn more about the limits HealthLake places on Bundles, see AWS HealthLake endpoints and quotas.

The following is an example of a Bundle operation containing multiple CRUD operations.

Example – Creating a Bundle request involving multiple CRUD operations.

To make a FHIR REST API request that performs multiple CRUD operations, you must make a POST request using your data store endpoint, and provide a JSON request body.

You can find your data store's endpoint in the HealthLake console under **Data Stores** or by using the DescribeFHIRDatastore operation in the *AWS HealthLake API Reference*.

POST Request

Make a POST request using your data store's endpoint. Use the next tab, **JSON Request Body** to see the required elements of the request body.

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
```

JSON Request Body

In the request body, you must provide the following key:value pairs along with any other FHIR resource-specific data about the individual CRUD requests. The first example shows a truncated JSON request body highlighting the required elements. The second example is shows a full JSON request body.

```
{
  "resourceType": "Bundle",
  "id": "bundle-batch-operation",
  "meta": {
    "lastUpdated": "2014-08-18T01:43:30Z"
  },
  "type": "batch", ## Required
  "entry": [
    {
      ## CRUD Transaction - 1
      "resource": {
        "resourceType": "Patient",
        ...
      },
      "request": { ## Required
        "method": "POST",
        "url": "Patient"
      }
    },
    {
      ## CRUD Transaction - 2
      "resource": {
        "resourceType": "Medication",
        ...
      },
      "request": { ## Required
        "method": "POST",
        "url": "Medication"
```

```
      }
    }
  ]
}
```

Here's a full example that shows creating a new `Patient` and `Medication` resource type.

```
{
  "resourceType": "Bundle",
  "id": "bundle-transaction",
  "meta": {
    "lastUpdated": "2014-08-18T01:43:30Z"
  },
  "type": "batch",
  "entry": [
    {
      "resource": {
        "resourceType": "Patient",
        "meta": {
          "lastUpdated": "2022-06-03T17:53:36.724Z"
        },
        "text": {
          "status": "generated",
          "div": "Some narrative"
        },
        "active": true,
        "name": [
          {
            "use": "official",
            "family": "Jackson",
            "given": [
              "Mateo",
              "James"
            ]
          }
        ],
        "gender": "male",
        "birthDate": "1974-12-25"
      },
      "request": {
        "method": "POST",
        "url": "Patient"
      }
```

```
    },
    {
      "resource": {
        "resourceType": "Medication",
        "id": "med0310",
        "contained": [
          {
            "resourceType": "Substance",
            "id": "sub03",
            "code": {
              "coding": [
                {
                  "system": "http://snomed.info/sct",
                  "code": "55452001",
                  "display": "Oxycodone (substance)"
                }
              ]
            }
          }
        ],
        "code": {
          "coding": [
            {
              "system": "http://snomed.info/sct",
              "code": "430127000",
              "display": "Oral Form Oxycodone (product)"
            }
          ]
        },
        "form": {
          "coding": [
            {
              "system": "http://snomed.info/sct",
              "code": "385055001",
              "display": "Tablet dose form (qualifier value)"
            }
          ]
        },
        "ingredient": [
          {
            "itemReference": {
              "reference": "#sub03"
            },
            "strength": {
```

```
              "numerator": {
                "value": 5,
                "system": "http://unitsofmeasure.org",
                "code": "mg"
              },
              "denominator": {
                "value": 1,
                "system": "http://terminology.hl7.org/CodeSystem/v3-
orderableDrugForm",
                "code": "TAB"
              }
            }
          }
        ]
      },
      "request": {
        "method": "POST",
        "url": "Medication"
      }
    }
  ]
}
```

JSON Response

To confirm the creation of the resources specified in the example bundle transaction, you get 201 Created HTTP status code for each included CRUD operation. When a CRUD operation fails you get 400 series HTTP status indicating why the individual request failed.

```
{
  "resourceType": "Bundle",
  "type": "batch-response",
  "timestamp": "2022-06-15T01:31:34.300+00:00",
  "entry": [
    {
      "response": {
        "status": "201",
        "location": "Patient/fd68ce38-ba30-4459-9eeb-476ad9f4f4ca",
        "lastModified": "2022-06-15T01:31:34.180+00:00"
      }
    },
    {
```

```
      "response": {
        "status": "201",
        "location": "Medication/5bf3b8cc-4076-4219-aba1-e2c53d7916f4",
        "lastModified": "2022-06-15T01:31:34.180+00:00"
      }
    }
  ]
}
```

Grouping resources as a Bundle resource type

To create a new Bundle resource type, you must specify `Bundle` in the FHIR REST API request and provide a valid JSON body containing the resources you want grouped together.

When Bundle is specified in the request URL, the contents of the JSON request body are saved in your HealthLake data store as-is. Therefore, no CRUD operations can be performed on the individual resource types. Bundles of this type are assigned a *single* new resource ID. Because the resources are saved as-is, you cannot make `GET` or `POST` requests on individual resources saved in the Bundle resource type.

> **Note**
>
> The HL7 FHIR R4 specification also supports grouping resources using Group, Composition, and List. When you create these resource types, the individual resources are not contained directly. Instead, they use the `Reference` element to point to the individual resources. Using these resources types therefore allow you to modify the individual resources contained within them.

To create a `Bundle` resource type, you must specify it in your `POST` request and provide a JSON enumerating the resources you want to be included.

Example – Creating a Bundle resource using a POST request

To create a `bundle` resource do the following

1. Format a FHIR REST API request as follows:

   `POST https://healthlake.`**`your-region`**`.amazonaws.com/datastore/`**`your-datastore-id`**`/r4/`**`Bundle`**

2. Provide JSON body which specifies the resources you want to group together. This example groups two patient resources.

```
{
  "resourceType": "Bundle",
  "id": "bundle-transaction",
  "meta": {
    "lastUpdated": "2018-03-11T11:22:16Z"
  },
  "type": "document",
  "entry": [
    {
      "resource": {
        "resourceType": "Patient",
        "name": [
          {
            "family": "Smith",
            "given": [
              "Jane"
            ]
          }
        ],
        "gender": "female",
        "address": [
          {
            "line": [
              "123 Main St."
            ],
            "city": "Anycity",
            "state": "Any State",
            "postalCode": "12345"
          }
        ]
      }
    },
    {
      "resource": {
        "resourceType": "Patient",
        "name": [
          {
            "family": "Jackson",
            "given": [
              "Mateo"
```

```
            ]
          }
        ],
        "gender": "male",
        "address": [
          {
            "line": [
              "1234 Main St."
            ],
            "city": "Anycity",
            "state": "Any State",
            "postalCode": "12345"
          }
        ]
      }
    }
  ]
}
```

Searching your HealthLake data store by using the FHIR REST API operations

HealthLake supports searching your data store by using the REST API operations provided as part of the FHIR standard. In this section, you will find examples of how to make GET and POST requests on multiple different resource types.

> **Note**
>
> For queries that involve Personally identifiable information (PII) or Protected Health Information (PHI) it's recommended to use POST requests. In a POST request, PII or PHI is added as part of the request body and is encrypted in transit.

The FHIR specification supports multiple search parameters types, but HealthLake supports only a subset. To learn more, see HealthLake supported search parameters.

Searching your data store using the FHIR REST API operations.

- HealthLake supported search parameters
  - Supported search parameter types

HealthLake supported search parameters

This topic provides details about supported search parameters, parameter types, modifiers, and comparators. Supported parameters are available for all FHIR resources, regardless of type.

Topics

Supported search parameter types

The following table shows the supported search parameter types in HealthLake.

Supported search parameters types

Search parameter	Description
_id	Resource id (not a full URL)
_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.
_security	Search on security labels applied to this resource.
_source	Search on where the resource comes from.
_text	Search on the narrative of the resource.
createdAt	Search on custom extension - createdAt.

Note

The following search parameters are only supported for datastores created after December 09, 2023 : _security, _source, _text, createdAt.

The following table shows examples of how to modify query strings based on specified data types for a given resource type. For clarity, special characters in the examples column have not been encoded. To make a successful query ensure that the query string has been properly encoded.

Search Parameter Types	Details	Examples

Number	Searches for a numerical value in a specified resource. Significant figures are observed.
	
	The number of significant digits are specific in by search parameter value, excluding leading zeros.
	
	Comparison prefixes are allowed.
		`[parameter]=100`
		
		`[parameter]=1e2`
		
		`[parameter]=lt100`

Date/DateTime	Searches for a specific date or time. The expected format is `yyyy-mm-ddThh:mm:ss[Z|(+|-)hh:mm]` but can vary.
	
	Accepts the following data types: `date`, `dateTime`, `instant`, `Period`, and `Timing`. For more details using these data types in searches, see date in the *FHIR Documentation Index*.
	
	Comparison prefixes are allowed.
		`[parameter]=eq2013-01-14`
		
		`[parameter]=gt2013-01-14T10:00`
		
		`[parameter]=ne2013-01-14`

String	Searches for a sequence of characters in a case-sensitive manner.	`[base]/Patient?given=eve`

Search Parameter Types	Details	Examples
	Supports both HumanName and Address types. For more details, see the HumanName data type entry and the Address data type entries in the *FHIR Documentation Index*. Advanced search is supported using :text modifiers.	[base]/Patient?given:contains=eve
Token	Searches for a close-to-exact match against a string of characters, often compared to a pair of medical code values. Case sensitivity is linked to the code system used when creating a query.Subsumption-based queries can help reduce issues linked to case sensitivity. For clarity the | has not been encoded.	[parameter]=[system]|[code] : Here [system] refers a coding system, and [code] refers to code value found within that specific system. [parameter]=[code] : Here your input will match either a code or a system. [parameter]=|[code] : Here your input will match a code, and the system property has no identifier.

Search Parameter Types	Details	Examples

Composite	Searches for multiple parameters within a single resource type, using the modifiers$ and , operation.
	
	Comparison prefixes are allowed.
		`/Patient?language=FR,NL&language=EN`
		
		`Observation?component-code-value-quantity=http://loinc.org|8480-6$lt60`
		
		`[base]/Group?characteristic-value=gender$mixed`

Quantity	Searches for a number, system, and code as values. A number is required, but system and code are optional. Based on the Quantity data type. For more details, see Quantity in the *FHIR Documentation Index*.
	
	Uses the following assumed syntax `[parameter]=[prefix][number]|[system]|[code]`
		`[base]/Observation?value-quantity=5.4|http://unitsofmeasure.org|mg`
		
		`[base]/Observation?value-quantity=5.4|http://unitsofmeasure.org|mg`
		
		`[base]/Observation?value-quantity=5.4|http://unitsofmeasure.org|mg`
		
		`[base]/Observation?value-quantity=le5.4|http://unitsofmeasure.org|mg`

Search Parameter Types	Details	Examples

Reference	Searches for references to other resources.	`[base]/Observation?subject=Patient/23`
		
		`test`

URI	Searches for a string of characters that unambiguously identifies a particular resource.	`[base]/ValueSet?url=http://acme.org/fhir/ValueSet/123`
Special	Searches based on integrated medical NLP extensions.	

Advanced search parameters supported by HealthLake

HealthLake supports the following advanced search parameters.

Name	Description	Example	Capability
`_include`	Used to request that additional resources be returned in a search request. It returns resources which are referenced by the target resource instance.	`Encounter?_include=Encounter:subject`	
`_revinclude`	Used to request that additional resources be returned in a search request. It returns resources that reference the primary resource instance.	`Patient?_id=` **`patient-identifier`** `&_revinclude=Encounter:patient`	

Name	Description	Example	Capability
_summary	Summary can be used to request a subset of the resource.	`Patient?_summary=text`	The following summary parameters are supported : `_summary=true` , `_summary=false` , `_summary=text` , `_summary=data` .
_elements	Request a specific set of elements to be returned as part of a resource in the search results.	`Patient?_elements=identifier,active,link`	
_total	Returns the number of resources that match the search parameters.	`Patient?_total=accurate`	Support `_total=accurate` , `_total=none` .
_sort	Indicate the sort order of the returned search results using a comma-separated list. The `-` prefix can be used for any sort rule in the comma-separated list to indicate decending order.	`Observation?_sort=status,-date`	Support sort by fields with types `Number, String, Quantity, Token, URI, Reference` . Sort by `Date` is only supported for datastores created after December 09, 2023. Support up to 5 sort rules.
_count	Control how many resources are returned per page of the search bundle.	`Patient?_count=100`	Maximum page size is 100.
chaining	Search elements of referenced resources. The `.` directs the chained search to the element within the referenced resource.	`DiagnosticReport?subject:Patient.name=peter`	

Name	Description	Example	Capability
reverse chainin (_has)	Search for a resource based on the elements of resources that refer to them.	Patient?_has:Observation:patient:code=1234-5	

_include

Using _include in a search query allows for additional specified FHIR resources to also be returned. Use _include to include resources that are linked forward.

Example – To use _include to find the patients or the group of patients who have been diagnosed with a cough

You would search on the Condition resource type specifying the diagnostic code for cough, and then using _include specify that you want the subject of that diagnosis returned too. In the Condition resource type subject refers to either the patient resource type or the group resource type.

For clarity, special characters in the example have not been encoded. To make a successful query ensure that the query string has been properly encoded.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Condition?code=49727002&_include=Condition:subject
```

_revinclude

Using _revinclude in a search query allows for additional specified FHIR resources to also be returned. Use _revinclude to include resources that are linked backwards.

Example – To use _revinclude to include related Encounter and Observation resource types linked to a specific Patient

To make this search, you would first define the individual Patient by specifying their identifier in the _id search parameter. Then you would specify additional FHIR resources using the structure Encounter:patient and Observation:patient.

For clarity, special characters in the example have not been encoded. To make a successful query ensure that the query string has been properly encoded.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?_id=patient-
identifier&_revinclude=Encounter:patient&_revinclude=Observation:patient
```

_summary

Using `_summary` in a search query allows user to request a subset of the FHIR resource. It can contain one of the following values: `true, text, data, false`. Any other values will be treated as invalid. The returned resources will be marked with `'SUBSETTED'` in meta.tag, to indicate that resources are incomplete.

- `true`: Return all supported elements that are marked as 'summary' in the base definition of the resource(s).
- `text`: Return only the 'text', 'id', 'meta' elements, and only top-level mandatory elements.
- `data`: Return all parts except the 'text' element.
- `false`: Return all parts of the resource(s)

In a single search request, `_summary=text` cannot be combined with `_include` or `_revinclude` search parameters.

Example – Get “text” element of Patient resources in a datastore.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?_summary=text
```

_elements

Using `_elements` in a search query allows for specific FHIR resource elements to be requested. The returned resources will be marked with `'SUBSETTED'` in meta.tag, to indicate that resources are incomplete.

The `_elements` parameter consists of a comma-separated list of base element names such as elements defined at the root level in the resource. Only elements that are listed are to be returned. If `_elements` parameter values contain invalid elements, server will ignore them and return mandatory elements and valid elements.

`_elements` will not be applicable to included resources(returned resources whose search mode is `include`).

In a single search request, `_elements` cannot be combined with `_summary` search parameters.

Example – Get “identifier”, “active”, “link” elements of Patient resources in your HealthLake datastore.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?_elements=identifier,active,link
```

`_total`

Using `_total` in a search query will return number of resources that match the requested search parameters. HealthLake will return the total number of matched resources(returned resources whose search mode is `match`) in the `Bundle.total` of search response.

`_total` supports the `accurate`, none parameter values. `_total=estimate` is not supported. Any other values will be treated as invalid. `_total` is not applicable to the included resources(returned resources whose search mode is `include`).

Example – Get the total number of Patient resources in a datastore:

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?_total=accurate
```

`_sort`

Using `_sort` in the search query arranges the results in a specific order. The results are ordered based on the comma-separated list of sort rules in priority order. The sort rules should be valid search parameters. Any other values will be treated as invalid.

In a single search request, you can use up to 5 sort search parameters. You can optionally use a `-` prefix to indicate descending order. Server will sort on ascending order by default.

The supported sort search parameter types are: `Number, String, Date, Quantity, Token, URI, Reference`. If a search parameter refers to an element that is nested, this search parameter is not supported for sort. For example, search on 'name' of resource type Patient refers to Patient.name element with HumanName data type is considered as nested. Thus, sort on Patient resources by 'name' is not supported.

Example – Get Patient resources in a datastore and sort them by birthdate in ascending order:

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?_sort=birthdate
```

_count

The parameter _count is defined as an instruction to the server regarding how many resources should be returned in a single page.

The maximum page size is 100. Any values greater than 100 is invalid. _count=0 is not supported.

Example – Search for the Patient resource and set search page size to 25:

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?_count=25
```

Chaining and Reverse Chaining(_has)

Chaining and reverse chaining in FHIR provide a more efficient and compact way to obtain interconnected data, reducing the need for multiple separate queries and making data retrieval more convenient for developers and users.

If any level of recursion return more than 100 results, HealthLake will return 4xx to protect datastore from being overloaded and causing multiple paginations.

Example – Chaining - Gets all DiagnosticReport which refer to a Patient where Patient name is peter.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
DiagnosticReport?subject:Patient.name=peter
```

Example – Reverse Chaining - Get Patient resources, where the patient resource is referred to by at least one Observation where the observation has a code of 1234, and where the Observation refers to the patient resource in the patient search parameter.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient?_has:Observation:patient:code=1234
```

Supported search modifiers

Search modifiers are used with string-based fields. All search modifiers in HealthLake use Boolean-based logic. For example, you could specify `:contains` to specify that larger string field should include a small string in order for it to be included in your search results.

Supported search modifiers

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

Supported search comparators

You can use search comparators to control the nature of the matching in a search. You can use comparators when searching on number, date, and quantity fields. The following table lists search comparators and their definitions that are supported by HealthLake.

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.

Search comparator	Description
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.

Search parameters not supported by HealthLake

For a full list of supported search parameters, see the FHIR search parameter registry. HealthLake supports all search parameters *except* for those listed in the table.

Unsupported search parameters

Bundle-composition	Location-near
Bundle-identifier	Consent-source-reference
Bundle-message	Contract-patient
Bundle-type	Resource-content

Bundle-timestamp	Resource-query

Search with POST examples

You can search a HealthLake data store by making POST requests. You can provide query parameters in either the URI or in a request body, but you cannot use both in a single request.

The examples in this topic follow that best practice.

> **Note**
>
> For queries that involve Personally identifiable information (PII) or Protected Health Information (PHI) it's recommended to use POST requests. In a POST request, PII or PHI is added as part of the request body and is encrypted in transit.

When making a POST request with a parameter in the request body, use `Content-Type: application/x-www-form-urlencoded` as part of the header.

This topic provides you with examples of how to search with POST by using the following resource types.

- **Age**: Age is not a defined resource type in FHIR. Instead, age is captured as a part of the Patient resource type. To search for a group of patients based on specific age or age range, use a the section called "Supported search comparators". For more details, see Resource type: Patient in the *FHIR Documentation Index*.
- **Condition**: This resource type stores details related to clinical concepts such as a diagnosis, situations, a clinical condition, and problems that have risen to a level of concern. To learn more, see Resource type: Condition in the *FHIR Documentation Index*. HealthLake creates new conditions based on documents found in the DocumentReference. These additions are excluded by default when making a POST request. To include them, you must specify a valid identifier for a condition resource in your search.
- **DocumentReference**:This resource type is supported by HealthLake. This resource type supports referencing documents of any type. To learn more, see Resource type: DocumentReference in the *FHIR Documentation Index*. HealthLake also provides integrated natural language processing (NLP) of documents found in the DocumentReference. To learn more, see Using automated

resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake.

- **Location**: This resource type includes both incidental locations (a place that is used for healthcare without prior designation or authorization) and dedicated, formally appointed locations. For more details, see Resource type: Location in the *FHIR Documentation Index*.
- **Observation**: Measurements and simple assertions made about a patient, device, or other subject. HealthLake creates new observation resources based on documents found in the DocumentReference resource. To learn more about how HealthLake creates new resources, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake. These additions are excluded by default when making a POST request. To include them, you must specify a valid identifier for an observation resource in your search. To learn more, see Resource type: Observation in the *FHIR Documentation Index*.

Each tab shows examples of how to search on the specified resource type. It includes an example of how to specify the request in the request body.

Age

Use the following to make a POST-based search request on the `Patient` resource type. This search uses the `eq` search comparator to search for individuals who were born in 1997.

You have to specify a request URL and a request body. Here is an example request URL.

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/_search
```

To specify the year 1997 in the search, you would add the following element to the request body.

```
birthdate=eq1997
```

JSON Response

When successful, you will get a `200` HTTP response code and a similar JSON response.

Condition

Using the following to make a POST request on the Condition resource type. This search finds locations in your HealthLake data store that contain the medical code 72892002.

You have to specify a request URL and a request body. Here is an example request URL.

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Condition/_search
```

To specify the medical code you want to search, you add this JSON element to the request body.

```
code=72892002
```

JSON Response

When successful, you will get a 200 HTTP response code. The following JSON response has been truncated for clarity.

```
{
 "resourceType": "Bundle",
 "type": "searchset",
 "entry": [{
   "resource": {
    "resourceType": "Condition",
    "id": "0063326c-6b42-4d13-af2f-1efe0a65f016",
    "meta": {
     "lastUpdated": "2022-08-23T00:22:49.681Z"
    },
    "clinicalStatus": {
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/condition-clinical",
      "code": "resolved"
     }]
    },
    "verificationStatus": {
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/condition-ver-status",
      "code": "confirmed"
     }]
    },
```

```
    "code": {
     "coding": [{
      "system": "http://snomed.info/sct",
      "code": "72892002",
      "display": "Normal pregnancy"
     }],
     "text": "Normal pregnancy"
    },
    "subject": {
     "reference": "Patient/5fc0070a-696a-4855-94a9-175f1c641a33"
    },
    "encounter": {
     "reference": "Encounter/44078ab9-7ac7-4731-9ac8-4b3ff21a7bdb"
    },
    "onsetDateTime": "2019-08-15T01:19:17-07:00",
    "abatementDateTime": "2020-03-26T01:19:17-07:00",
    "recordedDate": "2019-08-15T01:19:17-07:00"
   },
   "search": {
    "mode": "match"
   }
  },
  {
   "resource": {
    "resourceType": "Condition",
    "id": "d00afdb2-1d2c-44fe-9f3b-033c0fe751a3",
    "meta": {
     "lastUpdated": "2022-08-23T00:20:47.100Z"
    },
    "clinicalStatus": {
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/condition-clinical",
      "code": "resolved"
     }]
    },
    "verificationStatus": {
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/condition-ver-status",
      "code": "confirmed"
     }]
    },
    "code": {
     "coding": [{
      "system": "http://snomed.info/sct",
```

```
      "code": "72892002",
      "display": "Normal pregnancy"
     }],
     "text": "Normal pregnancy"
    },
    "subject": {
     "reference": "Patient/d0a5cd1e-8da7-41bd-9b2f-41eef45246e5"
    },
    "encounter": {
     "reference": "Encounter/73758e67-4aaf-4e80-982b-8821f0b6fdfb"
    },
    "onsetDateTime": "2019-06-13T20:37:40-07:00",
    "abatementDateTime": "2020-01-23T19:37:40-08:00",
    "recordedDate": "2019-06-13T20:37:40-07:00"
   },
   "search": {
    "mode": "match"
   }
  }
 ]
}
```

DocumentReference

To see the results of HealthLake's integrated natural language processing (NLP) when making a POST request on the `DocumentReference` resource type, format a request is as follows.

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
DocumentReference/_search
```

To specify the DocumentReference element you want to reference, see Search parameters. You'll specify those in the request body as JSON.

```
_lastUpdated=le2021-12-19&infer-icd10cm-entity-text-concept-score;=streptococcal|
0.6&infer-rxnorm-entity-text-concept-score=Amoxicillin|0.8
```

This query string uses multiple search parameters to search on Amazon Comprehend Medical API operations used to generate the integrated medical NLP results.

Location

Use the following to make a POST request on the `Location` resource type. This search finds locations in your HealthLake data store that contain the city name Boston as part of the address.

You must specify a request URL and a request body. Here is an example request URL.

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Location/_search
```

To specify Boston in the search, add the following element to the request body:

```
address=Boston
```

JSON Response

When successful, you will get a `200` HTTP response code. The JSON response has been truncated for clarity.

```
{
 "resourceType": "Bundle",
 "type": "searchset",
 "entry": [{
   "resource": {
    "resourceType": "Location",
    "id": "0a6903c7-25c5-4ae4-8354-be88f9c5f2ee",
    "meta": {
     "lastUpdated": "2022-08-23T00:24:24.570Z"
    },
    "status": "active",
    "name": "BRIGHAM AND WOMEN'S HOSPITAL",
    "telecom": [{
     "system": "phone",
     "value": "6177325500"
    }],
    "address": {
     "line": [
      "75 FRANCIS STREET"
     ],
     "city": "BOSTON",
     "state": "MA",
```

```
     "postalCode": "02115",
     "country": "US"
    },
    "position": {
     "longitude": -71.020173,
     "latitude": 42.33196
    },
    "managingOrganization": {
     "reference": "Organization/27379046-608b-32f0-9df7-8c833cf5d11d",
     "display": "BRIGHAM AND WOMEN'S HOSPITAL"
    }
   },
   "search": {
    "mode": "match"
   }
  },

  {
   "resource": {
    "resourceType": "Location",
    "id": "ca5e7f65-4eb5-4bff-9a6f-07bc80acf8d0",
    "meta": {
     "lastUpdated": "2022-08-23T00:20:47.100Z"
    },
    "status": "active",
    "name": "BETH ISRAEL DEACONESS MEDICAL CENTER",
    "telecom": [{
     "system": "phone",
     "value": "6176677000"
    }],
    "address": {
     "line": [
      "330 BROOKLINE AVENUE"
     ],
     "city": "BOSTON",
     "state": "MA",
     "postalCode": "02215",
     "country": "US"
    },
    "position": {
     "longitude": -71.020173,
     "latitude": 42.33196
    },
    "managingOrganization": {
```

```
     "reference": "Organization/cb6a50e0-af76-3758-99ad-3200ede03fff",
     "display": "BETH ISRAEL DEACONESS MEDICAL CENTER"
    }
   },
   "search": {
    "mode": "match"
   }
  }
 ]
}
```

Observation

Using the following to make a POST-based search request on the `Observation` resource type. This search uses the `value-concept` search parameter to look for medical code, `266919005`. This status indicates `Never smoker`.

You have to specify a request URL and a request body. Here is an example request URL.

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Observation/_search
```

To specify the status, `Never smoker` , set `value-concept=266919005` in the body of the JSON.

```
value-concept=266919005
```

JSON Response

When successful, you will get a `200` HTTP response code. The following JSON response has been truncated for clarity.

```
{
 "resourceType": "Bundle",
 "type": "searchset",
 "link": [{
  "relation": "next",
  "url": "https://healthlake.us-west-2.amazonaws.com/
datastore/3651c6d3c1e81e785adba06b710b52a9/r4/Observation?value-
concept=266919005&=AAMA-
EFRSURBSGlpcGIyN250ZG9WRXVnTTFOdmtxQk9Bb3Y0YjhVcVdUMGV0eVozNmdjQU9nRjRNUUtscjhCZ1NMUG84VGNqN
 }],
```

```
"entry": [{
  "resource": {
   "resourceType": "Observation",
   "id": "000038e0-71c6-4cc0-9c6c-50c8b1c53309",
   "meta": {
    "lastUpdated": "2022-11-03T01:02:38.981Z"
   },
   "status": "final",
   "category": [{
    "coding": [{
     "system": "http://terminology.hl7.org/CodeSystem/observation-category",
     "code": "survey",
     "display": "survey"
    }]
   }],
   "code": {
    "coding": [{
     "system": "http://loinc.org",
     "code": "72166-2",
     "display": "Tobacco smoking status NHIS"
    }],
    "text": "Tobacco smoking status NHIS"
   },
   "subject": {
    "reference": "Patient/598c9d7a-0494-448e-a81e-d50e3606e8db"
   },
   "encounter": {
    "reference": "Encounter/86bdee4a-2aa9-474a-b43f-6237cd68e512"
   },
   "effectiveDateTime": "2019-12-11T19:44:57-08:00",
   "issued": "2019-12-11T19:44:57.438-08:00",
   "valueCodeableConcept": {
    "coding": [{
     "system": "http://snomed.info/sct",
     "code": "266919005",
     "display": "Never smoker"
    }],
    "text": "Never smoker"
   }
  },
  "search": {
   "mode": "match"
  }
 },
```

```
{
 "resource": {
  "resourceType": "Observation",
  "id": "0c2f6260-e671-4cfd-ac3d-e75f073fa3cd",
  "meta": {
   "lastUpdated": "2022-11-03T01:05:21.488Z"
  },
  "status": "final",
  "category": [{
   "coding": [{
    "system": "http://terminology.hl7.org/CodeSystem/observation-category",
    "code": "survey",
    "display": "survey"
   }]
  }],
  "code": {
   "coding": [{
    "system": "http://loinc.org",
    "code": "72166-2",
    "display": "Tobacco smoking status NHIS"
   }],
   "text": "Tobacco smoking status NHIS"
  },
  "subject": {
   "reference": "Patient/89d9a9b7-9720-4881-a2ab-d7907544b26f"
  },
  "encounter": {
   "reference": "Encounter/8ebba7b0-fdfc-4ec1-a9aa-907cccf60925"
  },
  "effectiveDateTime": "2018-11-17T03:59:36-08:00",
  "issued": "2018-11-17T03:59:36.550-08:00",
  "valueCodeableConcept": {
   "coding": [{
    "system": "http://snomed.info/sct",
    "code": "266919005",
    "display": "Never smoker"
   }],
   "text": "Never smoker"
  }
 },
 "search": {
  "mode": "match"
 }
```

```
  }
 ]
}
```

Search with GET

You can search a HealthLake data store by making GET requests. HealthLake only supports providing query parameters as part of the URI, and not as part of a request body.

> **Note**
>
> For queries that involve Personally identifiable information (PII) or Protected Health Information (PHI) it's recommended to use POST requests. In a POST request, PII or PHI is added as part of the request body and is encrypted in transit.

The topic provides examples of how to search with GET using supported resource types in HealthLake.

- **Age**: Age is not a defined resource type in FHIR. Instead, age is captured as a part of the patient resource type. To search for a group of patients based on specific age or age range, you need to use a the section called "Supported search comparators". For more details, see Resource type: Patient in the *FHIR Documentation Index*.
- **Condition**: This resource type stores details related to clinical concepts such as a diagnosis, situations, a clinical condition, and problems that have risen to a level of concern. To learn more, see Resource type: Condition in the *FHIR Documentation Index*. HealthLake creates new conditions based on documents found in the DocumentReference. These additions are excluded by default when making a POST request. To include them, you must specify valid identifier for a condition resource in your search.
- **DocumentReference**:This resource type is supported by HealthLake. This resource type supports referencing documents of any type. To learn more, see Resource type: DocumentReference in the *FHIR Documentation Index*. HealthLake also provides integrated natural language processing (NLP) of documents found in the DocumentReference. To learn more, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake.

- **Location**: This resource type includes both incidental locations (a place that is used for healthcare without prior designation or authorization) and dedicated, formally appointed locations For more details, see Resource type: Location in the *FHIR Documentation Index*.
- **Observation**: Measurements and simple assertions made about a patient, device, or other subject. HealthLake creates new observation resources based on documents found in the DocumentReference resource. To learn more about how HealthLake create new resources, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake. These additions are excluded by default when making a POST request. To include them, you must specify a valid identifier for an observation resource in your search. To learn more, see Resource type: Observation in the *FHIR Documentation Index*.

Each tab shows an example of how to search on the specified resource type. It includes an example of how to specify the request in the URI, and the related JSON response.

Age

Use the following to make a GET-based search request on the `Patient` resource type. This search uses the `eq` search comparator to search for individuals who were born in 1997.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4//
Patient?birthdate=eq1997
```

JSON Response

When successful, you will get a `200` HTTP response code.

Condition

Use the following to make a GET request on the `Condition` resource type. This search finds locations in your HealthLake data store that contain the medical code `72892002`.

You have to specify a request URL and a request body. Here is an example request URL.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Condition?code=72892002
```

JSON Response

When successful, you will get a 200 HTTP response code. The following JSON response has been truncated for clarity.


```
{
 "resourceType": "Bundle",
 "type": "searchset",
 "entry": [{
   "resource": {
    "resourceType": "Condition",
    "id": "0063326c-6b42-4d13-af2f-1efe0a65f016",
    "meta": {
     "lastUpdated": "2022-08-23T00:22:49.681Z"
    },
    "clinicalStatus": {
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/condition-clinical",
      "code": "resolved"
     }]
    },
    "verificationStatus": {
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/condition-ver-status",
      "code": "confirmed"
     }]
    },
    "code": {
     "coding": [{
      "system": "http://snomed.info/sct",
      "code": "72892002",
      "display": "Normal pregnancy"
     }],
     "text": "Normal pregnancy"
    },
    "subject": {
     "reference": "Patient/5fc0070a-696a-4855-94a9-175f1c641a33"
    },
    "encounter": {
     "reference": "Encounter/44078ab9-7ac7-4731-9ac8-4b3ff21a7bdb"
    },
    "onsetDateTime": "2019-08-15T01:19:17-07:00",
    "abatementDateTime": "2020-03-26T01:19:17-07:00",
    "recordedDate": "2019-08-15T01:19:17-07:00"
   },
```

```
  "search": {
   "mode": "match"
  }
 },
 {
  "resource": {
   "resourceType": "Condition",
   "id": "d00afdb2-1d2c-44fe-9f3b-033c0fe751a3",
   "meta": {
    "lastUpdated": "2022-08-23T00:20:47.100Z"
   },
   "clinicalStatus": {
    "coding": [{
     "system": "http://terminology.hl7.org/CodeSystem/condition-clinical",
     "code": "resolved"
    }]
   },
   "verificationStatus": {
    "coding": [{
     "system": "http://terminology.hl7.org/CodeSystem/condition-ver-status",
     "code": "confirmed"
    }]
   },
   "code": {
    "coding": [{
     "system": "http://snomed.info/sct",
     "code": "72892002",
     "display": "Normal pregnancy"
    }],
    "text": "Normal pregnancy"
   },
   "subject": {
    "reference": "Patient/d0a5cd1e-8da7-41bd-9b2f-41eef45246e5"
   },
   "encounter": {
    "reference": "Encounter/73758e67-4aaf-4e80-982b-8821f0b6fdfb"
   },
   "onsetDateTime": "2019-06-13T20:37:40-07:00",
   "abatementDateTime": "2020-01-23T19:37:40-08:00",
   "recordedDate": "2019-06-13T20:37:40-07:00"
  },
  "search": {
   "mode": "match"
  }
```

```
  }
 ]
}
```

DocumentationReference

This example shows how to create a search request on the `DocumentReference` resource type for patients with a streptococcal diagnosis and who have also been prescribed amoxicillin.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-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
```

When successful you will get 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"
                                }
                              ]
                            }
```

Location

Use the following to make a GET request on the `Location` resource type. This search finds locations in your HealthLake data store that contain the city name Boston as part of the address.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4//
Location?address=boston
```

JSON Response

When successful, you will get a `200` HTTP response code. The JSON response has been truncated for clarity.

```
{
    "resourceType": "Bundle",
    "type": "searchset",
    "entry": [
```

```
        {
            "resource": {
                "resourceType": "Location",
                "id": "0a6903c7-25c5-4ae4-8354-be88f9c5f2ee",
                "meta": {
                    "lastUpdated": "2022-08-23T00:24:24.570Z"
                },
                "status": "active",
                "name": "BRIGHAM AND WOMEN'S HOSPITAL",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6177325500"
                    }
                ],
                "address": {
                    "line": [
                        "75 FRANCIS STREET"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02115",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference":
"Organization/27379046-608b-32f0-9df7-8c833cf5d11d",
                    "display": "BRIGHAM AND WOMEN'S HOSPITAL"
                }
            },
            "search": {
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
                "id": "3cc3ad99-e0ff-48b4-b277-052abfc41058",
                "meta": {
                    "lastUpdated": "2022-08-23T00:19:37.029Z"
```

```
                },
                "status": "active",
                "name": "NEW ENGLAND BAPTIST HOSPITAL",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6177545800"
                    }
                ],
                "address": {
                    "line": [
                        "125 PARKER HILL AVENUE"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02120",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference": "Organization/9a7149fa-49fc-3c87-b935-
d29c55808717",
                    "display": "NEW ENGLAND BAPTIST HOSPITAL"
                }
            },
            "search": {
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
                "id": "3f956715-3890-4235-85be-3fba5e3488ee",
                "meta": {
                    "lastUpdated": "2022-08-23T00:23:38.981Z"
                },
                "status": "active",
                "name": "MASSACHUSETTS GENERAL HOSPITAL",
                "telecom": [
                    {
                        "system": "phone",
```

```
                        "value": "6177262000"
                    }
                ],
                "address": {
                    "line": [
                        "55 FRUIT STREET"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02114",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference": "Organization/d78e84ec-30aa-3bba-a33a-
f29a3a454662",
                    "display": "MASSACHUSETTS GENERAL HOSPITAL"
                }
            },
            "search": {
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
                "id": "6cc07b51-7287-443c-b772-c864f7831e13",
                "meta": {
                    "lastUpdated": "2022-08-23T00:21:11.045Z"
                },
                "status": "active",
                "name": "TUFTS MEDICAL CENTER",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6176365000"
                    }
                ],
                "address": {
                    "line": [
                        "800 WASHINGTON STREET"
```

```
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02111",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference": "Organization/b7175ab4-
bde5-3848-891b-579bccb77c7c",
                    "display": "TUFTS MEDICAL CENTER"
                }
            },
            "search": {
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
                "id": "8101300f-f685-49e7-b428-43b7855c39ee",
                "meta": {
                    "lastUpdated": "2022-08-23T00:22:06.474Z"
                },
                "status": "active",
                "name": "BOSTON CHILDREN'S HOSPITAL",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6177356000"
                    }
                ],
                "address": {
                    "line": [
                        "300 LONGWOOD AVENUE"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02115",
                    "country": "US"
                },
```

```
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference": "Organization/d7b11827-25f2-350b-
bcd8-939fc59851b0",
                    "display": "BOSTON CHILDREN'S HOSPITAL"
                }
            },
            "search": {
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
                "id": "8b7641d3-6997-48bb-bd60-23e35dfaae9d",
                "meta": {
                    "lastUpdated": "2022-08-23T00:20:47.099Z"
                },
                "status": "active",
                "name": "BRIGHAM AND WOMEN'S FAULKNER HOSPITAL",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6179837000"
                    }
                ],
                "address": {
                    "line": [
                        "1153 CENTRE STREET"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02130",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
```

```
                    "reference": "Organization/d733d4a9-080d-3593-
b910-2366e652b7ea",
                    "display": "BRIGHAM AND WOMEN'S FAULKNER HOSPITAL"
                }
            },
            "search": {
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
                "id": "998ef80b-7b58-4dc3-99ac-c440ec9e282d",
                "meta": {
                    "lastUpdated": "2022-08-23T00:21:11.046Z"
                },
                "status": "active",
                "name": "BRIGHAM AND WOMEN'S FAULKNER HOSPITAL",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6179837000"
                    }
                ],
                "address": {
                    "line": [
                        "1153 CENTRE STREET"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02130",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference": "Organization/d733d4a9-080d-3593-
b910-2366e652b7ea",
                    "display": "BRIGHAM AND WOMEN'S FAULKNER HOSPITAL"
                }
            },
            "search": {
```

```
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
                "id": "c454bed3-7013-4376-81cf-4f49342f1402",
                "meta": {
                    "lastUpdated": "2022-08-23T00:24:24.573Z"
                },
                "status": "active",
                "name": "MASSACHUSETTS GENERAL HOSPITAL",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6177262000"
                    }
                ],
                "address": {
                    "line": [
                        "55 FRUIT STREET"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02114",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference": "Organization/d78e84ec-30aa-3bba-a33a-
f29a3a454662",
                    "display": "MASSACHUSETTS GENERAL HOSPITAL"
                }
            },
            "search": {
                "mode": "match"
            }
        },
        {
            "resource": {
                "resourceType": "Location",
```

```
                "id": "ca5e7f65-4eb5-4bff-9a6f-07bc80acf8d0",
                "meta": {
                    "lastUpdated": "2022-08-23T00:20:47.100Z"
                },
                "status": "active",
                "name": "BETH ISRAEL DEACONESS MEDICAL CENTER",
                "telecom": [
                    {
                        "system": "phone",
                        "value": "6176677000"
                    }
                ],
                "address": {
                    "line": [
                        "330 BROOKLINE AVENUE"
                    ],
                    "city": "BOSTON",
                    "state": "MA",
                    "postalCode": "02215",
                    "country": "US"
                },
                "position": {
                    "longitude": -71.020173,
                    "latitude": 42.33196
                },
                "managingOrganization": {
                    "reference": "Organization/cb6a50e0-
af76-3758-99ad-3200ede03fff",
                    "display": "BETH ISRAEL DEACONESS MEDICAL CENTER"
                }
            },
            "search": {
                "mode": "match"
            }
        }
    ]
}
```

Observation

Use the following to make a `GET`-based search request on the `Observation` resource type. This search uses the `value-concept` search parameter to look for medical code, `266919005`. This status indicates `Never smoker`.

You have to specify a request URL and a query string. Here is an example request URL.

```
POST https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Observation?value-concept=266919005
```

To specify the status, `Never smoker`, set `value-concept=266919005` as the query string.

JSON Response

When successful, you will get a `200` HTTP response code. The following JSON response has been truncated for clarity.

```
{
 "resourceType": "Bundle",
 "type": "searchset",
 "link": [{
  "relation": "next",
  "url": "https://healthlake.us-west-2.amazonaws.com/
datastore/3651c6d3c1e81e785adba06b710b52a9/r4/Observation?value-
concept=266919005&=AAMA-
EFRSURBSGlpcGIyN250ZG9WRXVnTTFOdmtxQk9Bb3Y0YjhVcVdUMGV0eVozNmdjQU9nRjRNUUtscjhCZ1NMUG84VGNqN
 }],
 "entry": [{
   "resource": {
    "resourceType": "Observation",
    "id": "000038e0-71c6-4cc0-9c6c-50c8b1c53309",
    "meta": {
     "lastUpdated": "2022-11-03T01:02:38.981Z"
    },
    "status": "final",
    "category": [{
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/observation-category",
      "code": "survey",
      "display": "survey"
     }]
    }],
    "code": {
     "coding": [{
      "system": "http://loinc.org",
      "code": "72166-2",
      "display": "Tobacco smoking status NHIS"
     }],
```

```
     "text": "Tobacco smoking status NHIS"
    },
    "subject": {
     "reference": "Patient/598c9d7a-0494-448e-a81e-d50e3606e8db"
    },
    "encounter": {
     "reference": "Encounter/86bdee4a-2aa9-474a-b43f-6237cd68e512"
    },
    "effectiveDateTime": "2019-12-11T19:44:57-08:00",
    "issued": "2019-12-11T19:44:57.438-08:00",
    "valueCodeableConcept": {
     "coding": [{
      "system": "http://snomed.info/sct",
      "code": "266919005",
      "display": "Never smoker"
     }],
     "text": "Never smoker"
    }
   },
   "search": {
    "mode": "match"
   }
  },

  {
   "resource": {
    "resourceType": "Observation",
    "id": "0c2f6260-e671-4cfd-ac3d-e75f073fa3cd",
    "meta": {
     "lastUpdated": "2022-11-03T01:05:21.488Z"
    },
    "status": "final",
    "category": [{
     "coding": [{
      "system": "http://terminology.hl7.org/CodeSystem/observation-category",
      "code": "survey",
      "display": "survey"
     }]
    }],
    "code": {
     "coding": [{
      "system": "http://loinc.org",
      "code": "72166-2",
      "display": "Tobacco smoking status NHIS"
```

```
      }],
      "text": "Tobacco smoking status NHIS"
     },
     "subject": {
      "reference": "Patient/89d9a9b7-9720-4881-a2ab-d7907544b26f"
     },
     "encounter": {
      "reference": "Encounter/8ebba7b0-fdfc-4ec1-a9aa-907cccf60925"
     },
     "effectiveDateTime": "2018-11-17T03:59:36-08:00",
     "issued": "2018-11-17T03:59:36.550-08:00",
     "valueCodeableConcept": {
      "coding": [{
       "system": "http://snomed.info/sct",
       "code": "266919005",
       "display": "Never smoker"
      }],
      "text": "Never smoker"
     }
    },
    "search": {
     "mode": "match"
    }
   }
  ]
}
```

Extended FHIR operations on HealthLake data stores

These topics describe how to perform FHIR operations on your HealthLake data store using the FHIR REST API.

Topics

- Get Patient Data with Patient $everything
- Exporting data from your HealthLake data store using $export

Get Patient Data with Patient $everything

The Patient $everything operation is used to query a FHIR Patient resource along with any other resources that are related to that patient. This operation can be used to provide a patient with

access to their entire record or for a provider to perform a bulk data download related to a patient. HealthLake supports $everything for a specific patient id.

> **Note**
>
> The Patient $everything operation is currently supported on data stores created after Feb 27, 2024.

Get all resources related to a patient

Patient $everything is a REST API operation that can be invoked as shown in the examples below.

GET Request

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/patient-id/$everything
```

> **Note**
>
> Resources in response are sorted by resource type and resource id.
> Response is always populated with Bundle.total.

Patient $everything Parameters

HealthLake supports the following query parameters

Parameter	Details
start	Get all patient data after a specified start date.
end	Get all patient data before a specified end date.
since	Get all patient data updated after a specified date.
_type	Get patient data for specific resource types.

Parameter	Details
_count	Get patient data and specify page size.

Example - Get all patient data after a specified start date

Patient $everything can use the `start` filter to only query data after a specific date.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/patient-id/$everything?start=2024-03-15T00:00:00.000Z
```

Example - Get all patient data before a specified end date

Patient $everything can use the end filter to only query data before a specific date.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/patient-id/$everything?end=2024-03-15T00:00:00.000Z
```

Example - Get all patient data updated after a specified date

Patient $everything can use the `since` filter to only query data updated after a specific date.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/patient-id/$everything?since=2024-03-15T00:00:00.000Z
```

Example - Get patient data for specific resource types

Patient $everything can use the `_type` filter to specify specific resource types to be included in the response. Multiple resource types can be specified in a comma separated list.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/patient-id/$everything?_type=Observation,Condition
```

Example - Get patient data and specify page size

Patient $everything can use the `_count` to set the page size.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/patient-id/$everything?_count=15
```

Patient $everything `start` and `end` attributes

HealthLake supports the following resource attributes for the start and end query parameters.

Resource	Resource Element
Account	Account.servicePeriod.start
AdverseEvent	AdverseEvent.date
AllergyIntolerance	AllergyIntolerance.recordedDate
Appointment	Appointment.start
AppointmentResponse	AppointmentResponse.start
AuditEvent	AuditEvent.period.start
Basic	Basic.created
BodyStructure	NO_DATE
CarePlan	CarePlan.period.start
CareTeam	CareTeam.period.start
ChargeItem	ChargeItem.occurrenceDateTime, ChargeItem.occurrencePeriod.start, ChargeItem.occurrenceTiming.event
Claim	Claim.billablePeriod.start

Resource	Resource Element
ClaimResponse	ClaimResponse.created
ClinicalImpression	ClinicalImpression.date
Communication	Communication.sent
CommunicationRequest	CommunicationRequest.occurrenceDateTime, CommunicationRequest.occurrencePeriod.start
Composition	Composition.date
Condition	Condition.recordedDate
Consent	Consent.dateTime
Coverage	Coverage.period.start
CoverageEligibilityRequest	CoverageEligibilityRequest.created
CoverageEligibilityResponse	CoverageEligibilityResponse.created
DetectedIssue	DetectedIssue.identified
DeviceRequest	DeviceRequest.authoredOn
DeviceUseStatement	DeviceUseStatement.recordedOn

Resource	Resource Element
DiagnosticReport	DiagnosticReport.effective
DocumentManifest	DocumentManifest.created
DocumentReference	DocumentReference.context.period.start
Encounter	Encounter.period.start
EnrollmentRequest	EnrollmentRequest.created
EpisodeOfCare	EpisodeOfCare.period.start
ExplanationOfBenefit	ExplanationOfBenefit.billablePeriod.start
FamilyMemberHistory	NO_DATE
Flag	Flag.period.start
Goal	Goal.statusDate
Group	NO_DATE
ImagingStudy	ImagingStudy.started
Immunization	Immunization.recorded

Resource	Resource Element
ImmunizationEvaluation	ImmunizationEvaluation.date
ImmunizationRecommendation	ImmunizationRecommendation.date
Invoice	Invoice.date
List	List.date
MeasureReport	MeasureReport.period.start
Media	Media.issued
MedicationAdministration	MedicationAdministration.effective
MedicationDispense	MedicationDispense.whenPrepared
MedicationRequest	MedicationRequest.authoredOn
MedicationStatement	MedicationStatement.dateAsserted
MolecularSequence	NO_DATE
NutritionOrder	NutritionOrder.dateTime

Resource	Resource Element
Observation	Observation.effective
Patient	NO_DATE
Person	NO_DATE
Procedure	Procedure.performed
Provenance	Provenance.occurredPeriod.start, Provenance.occurredDateTime
QuestionnaireResponse	QuestionnaireResponse.authored
RelatedPerson	NO_DATE
RequestGroup	RequestGroup.authoredOn
ResearchSubject	ResearchSubject.period
RiskAssessment	RiskAssessment.occurrenceDateTime, RiskAssessment.occurrencePeriod.start
Schedule	Schedule.planningHorizon
ServiceRequest	ServiceRequest.authoredOn
Specimen	Specimen.receivedTime
SupplyDelivery	SupplyDelivery.occurrenceDateTime, SupplyDelivery.occurrencePeriod.start, SupplyDelivery.occurrenceTiming.event

Resource	Resource Element
SupplyRequest	SupplyRequest.authoredOn
VisionPrescription	VisionPrescription.dateWritten

Exporting data from your HealthLake data store using $export

Important

HealthLake data stores created prior to June 1, 2023 only support FHIR REST API based export job requests for system-wide exports.
HealthLake data stores created prior to June 1, 2023 do not support getting the status of an export using a GET request on a data store's endpoint.

To make an export request using the FHIR REST API specify `$export` as part of the POST request, and include request parameters in the body of your request. According to the FHIR specification, the FHIR server must support GET requests, and can support POST requests. In order to support additional parameters, a body is needed to start the export, therefore HealthLake supports POST requests.

All export requests you make using the FHIR REST API are returned in `ndjson` format and exported to an Amazon S3 bucket. Each S3 object will contain only a single FHIR resource type.

You can make a single export request for each AWS account at a time. To learn more about the Service Quotas associated with HealthLake, see AWS HealthLake endpoints and quotas.

To learn more about making an export requesting using the FHIR REST API, see Exporting data from your data store by using the FHIR REST API.

Query AWS HealthLake data stores using SQL in Amazon Athena

> **Note**
>
> After February 20, 2023, HealthLake data stores do *not* use integrated natural language processing (NLP) by default. If you are interested in turning on this feature on your data store, see How do I turn on HealthLake's integrated natural language processing feature? in the Troubleshooting chapter.

When you create a HealthLake data store the highly nested FHIR data structure is ingested into Amazon Athena, and automatically transformed into Iceberg tables queryable with SQL. Granting access to this new resource is managed using AWS Lake Formation. Each FHIR resource type is represented as an individual table in Athena.

> **Important**
>
> For data stores created before November, 14, 2022, you must migrate your existing data store to a new one to query it using SQL. For help, see Migrating an existing data store to use Amazon Athena.

To create a HealthLake data store, you must add additional IAM policies and a service role to your IAM user or role that is a HealthLake administrator. To learn more about these changes, see Getting started with AWS HealthLake.

HealthLake data stores are ingested into Athena as Iceberg tables. To learn more about how Iceberg tables function in Athena, see Using Iceberg tables in the *Athena User Guide*.

HealthLake supports READ operations of your HealthLake data stores data stores in Athena. To learn more about Create, Read, Update, and Delete (CRUD) operations using the FHIR REST API operations, see Managing and searching resources in AWS HealthLake by using FHIR REST API operations to learn more about how CRUD operations affect your data in Athena.

The topics in this chapter describe how to connect your HealthLake data store to Athena, how to query it using SQL, and how to connect results with other AWS services for further analysis.

Contents

Connecting your data store to Amazon Athena

Important

After November, 14, 2022, the IAM requirements to access HealthLake changed. To both create data stores and to grant access to them in Athena, you must have the `AWSLakeFormationDataAdmin` managed policy added to your IAM user, group or role. You can use the `AWSLakeFormationDataAdmin` policy to create data lake administrators and grant access to data stores in Athena.

This topic outlines the necessary steps to create an Athena user, group or role, and grant them access to FHIR resources found in a HealthLake data store.

- Granting a user, group, or role access to a HealthLake data store (AWS Lake Formation Console)
- Setting up an Athena account

Granting a user, group, or role access to a HealthLake data store (AWS Lake Formation Console)

Persona: HealthLake administrator

The HealthLake administrator persona is a data lake administrator in AWS Lake Formation. They grant access to HealthLake data stores in Lake Formation.

For each data store created, there are two entries visible in the AWS Lake Formation console. One entry is a *resource link*. Resource link names are always displayed in *italics*. Each resource link is displayed with the name and owner of its linked shared resource. For all HealthLake data stores, the shared resource owner is the HealthLake service account. The other entry is the HealthLake data store in the HealthLake service account. The steps in this procedure use the data store that is the resource link.

To learn more about resource links, see How resource links work in Lake Formation in the *AWS Lake Formation Developer Guide*.

For a user, group, or role to be able to query data in Athena, you must grant **Describe** permission on the resource database. Then, you must grant **Select** and **Describe** on the tables.

STEP 1: To grant DESCRIBE permissions on a HealthLake data store resource link database

1. Open the AWS Lake Formation console: https://console.aws.amazon.com/lakeformation/
2. In the primary navigation bar, choose **Databases**.
3. On the **Databases** page, choose the radio button next to the name of the data store that is in italics.
4. Choose **Actions** (▼).
5. Choose **Grant**.
6. On the **Grant data permissions** page, under **Principals**, choose **IAM users or roles**.
7. Under **IAM users or roles**, use the **down arrow** (▼), or search for the IAM user, role, or group that you want to be able to make queries on in Athena.
8. Under **LF-Tags or catalog resources** card, choose the **Named data catalog resources** option.
9. Under **Databases**, use the **down arrow** (▼) to choose the HealthLake data store database that you want to share access to.
10. In the **Resource link permissions** card, under **Resource link permissions**, choose **Describe**.

When the grant is successful, the **Grant permission success** banner appears. To view the permission you just granted, choose **Data lake permissions**. Find the user, group, and role in the table. Under the **Permissions** column, you will see **Describe** listed.

Now you must use **Grant on target** to grant **Select** and **Describe** on all tables in the database.

STEP 2: Grant access to all tables in a HealthLake data store resource link

1. Open the AWS Lake Formation console: https://console.aws.amazon.com/lakeformation/
2. In the primary navigation bar, choose **Databases**.
3. On the **Databases** page, choose the radio button next to the name of the data store that is in italics.
4. Choose **Actions** (▾).
5. Choose **Grant on target**.
6. On the **Grant data permissions** page, under **Principals**, choose **IAM users or roles**.
7. Under **IAM users or roles**, use the **down arrow** (▾) or search for the IAM user, group, or role that you want to be able to make queries on in Athena.
8. Under **LF-Tags or catalog resources** card, choose the **Named data catalog resources** option.
9. Under **Databases**, use the **down arrow** (▾) to choose the HealthLake data store database that you want to grant access to.
10. Under **Tables**, choose **All tables** to share all tables with a HealthLake user.
11. In the **Table permissions** card, under **Table permissions**, choose **Describe** and **Select**.
12. Choose **Grant**.

After choosing grant,a **Grant permissions success** banner appears. The specified user can now make queries on a HealthLake data store in Athena.

Getting started with Athena

> **HealthLake user**
>
> The HealthLake user will use the Athena console, AWS CLI, or AWS SDKs to query a HealthLake data store shared with them by the HealthLake administrator.

To query a data store using Athena, you must do the following three things.

- Grant the IAM user or role access to the HealthLake data store via Lake Formation. To learn more, see Granting a user, group, or role access to a HealthLake data store (AWS Lake Formation Console).
- Create a workgroup for your HealthLake data store.

- Designate an Amazon S3 bucket to store your query results.

To get started with Athena, add the **AmazonAthenaFullAccess** and **AmazonS3FullAccess** AWS managed policies to your user, group or role. Using an AWS managed policy is great way to get started using a new service. Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they are available for use by all AWS customers. When you set permissions with IAM policies, grant only the permissions required to perform a task. To learn more about IAM and applying least-privilege, see Apply least-privilege permissions in the *IAM User Guide*.

> **Important**
>
> To query a HealthLake data store in Athena, you must use **Athena engine version 3**.

Workgroups are resources, and therefore you can use IAM-based policies to control access to specific workgroups. To learn more, see Using workgroups to control query access and costs in the *Athena User Guide*.

To learn more about setting up workgroups, see https://docs.aws.amazon.com/athena/latest/ug/workgroups-procedure.html in the *Athena User Guide*.

> **Note**
>
> The region your Amazon S3 bucket is in and the Athena console must match.

Before you can run a query, a query result bucket location in Amazon S3 must be specified, or you must use a workgroup that has specified a bucket and whose configuration overrides client settings. Output files are saved automatically for every query that runs.

For more details on specifying query result locations in the Athena console, see Specifying a query result location using the Athena console in the *Amazon Athena User Guide*.

To see examples of how to query your HealthLake data store in Athena, see Query your HealthLake data store using SQL.

Query your HealthLake data store using SQL

> **Note**
>
> After February 20, 2023, HealthLake data stores do *not* use integrated natural language processing (NLP) by default. If you are interested in turning on this feature on your data store, see How do I turn on HealthLake's integrated natural language processing feature? in the Troubleshooting chapter.
> All examples in this topic use fictionalized data created using Synthea. To learn more about creating a data store preloaded with Synthea data, see Getting started with AWS HealthLake.

When you import your HealthLake data store into Athena, each resource type from your HealthLake data store is converted into a table. These tables can be queried individually or as group using SQL-based queries. Because of the structure of data stores, your data is imported into Athena as multiple different data types. To learn more about creating SQL queries that can access these data types, see Querying arrays with complex types and nested structures in the *Amazon Athena User Guide*.

For each element in a resource type, the FHIR specification defines a cardinality. The cardinality of an element defines the lower and upper bounds of how many times this element can appear. When constructing a SQL query, you must take this into account. For example, let's look at some elements in Resource type: Patient.

- **Element: Name** The FHIR specification sets the cardinality as `0..*`.

  The element is captured as an array.

```
[{
 id = null,
 extension = null,
 use = official,
 _use = null,
 text = null,
 _text = null,
 family = Wolf938,
 _family = null,
 given = [Noel608],
 _given = null,
```

```
 prefix = null,
 _prefix = null,
 suffix = null,
 _suffix = null,
 period = null
}]
```

In Athena, to see how a resource type has been ingested, search for it under **Tables and views**. To access elements in this array, you can use dot notation. Here's a simple example that would access the values for `given` and `family`.

```
SELECT
    name[1].given as FirstName,
    name[1].family as LastName
FROM Patient
```

- **Element: MaritalStatus** The FHIR specification sets the cardinality as `0..1`.

  This element is captured as JSON.

```
{
 id = null,
 extension = null,
 coding = [
  {
   id = null,
   extension = null,
   system = http: //terminology.hl7.org/CodeSystem/v3-MaritalStatus,
    _system = null,
   version = null,
   _version = null,
   code = S,
   _code = null,
   display = Never Married,
   _display = null,
   userSelected = null,
   _userSelected = null
  }

 ],
 text = Never Married,
 _text = null
```

```
}
```

In Athena, to see how a resource type has been ingested, search for it under **Tables and views**. To access key-value pairs in the JSON, you can use dot notation. Because it isn't an array, no array index is required. Here's a simple example that would access the value for `text`.

```
SELECT
    maritalstatus.text as MaritalStatus
FROM Patient
```

To learn more about accessing and searching JSON, see Querying JSON in the *Athena User Guide*.

Athena Data Manipulation Language (DML) query statements are based on Trino. Athena does not support all of Trino's features, and there are *significant* differences. To learn more, see DML queries, functions, and operators in the *Amazon Athena User Guide*.

Furthermore, Athena supports multiple data types that you may encounter when creating queries of your HealthLake data store. To learn more about data types in Athena, see Data types in Amazon Athena in the *Amazon Athena User Guide*.

To learn more about how SQL queries work in Athena, see SQL reference for Amazon Athena in the *Amazon Athena User Guide*.

Each tab shows examples of how to search on the specified resource types and associated elements using Athena.

Element: Extension

The element `extension` is used to create custom fields in a data store.

This example shows you how to access the features of the `extension` element found in the `Patient` resource type.

When your HealthLake data store is imported into Athena, the elements of a resource type are parsed differently. Because the structure of the `element` is variable, it cannot be fully specified in the schema. To handle that variability, the elements inside the array are passed as strings.

In the table description of `Patient`, you can see the element `extension` described as `array<string>`, which means you can access the elements of array by using an index value. To access the elements of the string, however, you must use `json_extract`.

Here is a single entry from the `extension` element found in the patient table.

```
[{
  "valueString": "Kerry175 Cummerata161",
  "url": "http://hl7.org/fhir/StructureDefinition/patient-mothersMaidenName"
 },
 {
  "valueAddress": {
   "country": "DE",
   "city": "Hamburg",
   "state": "Hamburg"
  },
  "url": "http://hl7.org/fhir/StructureDefinition/patient-birthPlace"
 },
 {
  "valueDecimal": 0.0,
  "url": "http://synthetichealth.github.io/synthea/disability-adjusted-life-years"
 },
 {
  "valueDecimal": 5.0,
  "url": "http://synthetichealth.github.io/synthea/quality-adjusted-life-years"
 }
]
```

Even though this is valid JSON, Athena treats it as a string.

This SQL query example demonstrates how you can create a table that contains the `patient-mothersMaidenName` and `patient-birthPlace` elements. To access these elements, you need to use different array indices and `json_extract`.

```
SELECT
    extension[1],
    json_extract(extension[1], '$.valueString') AS MothersMaidenName,
    extension[2],
    json_extract(extension[2], '$.valueAddress.city') AS birthPlace
FROM patient
```

To learn more about queries that involve JSON, see Extracting data from JSON in the *Amazon Athena User Guide*.

Element: birthDate (Age)

Age is *not* an element of the Patient resource type in FHIR. Here are two examples for searches that filter based on age.

Because age is not an element, we use the `birthDate` for the SQL queries. To see how an element has been ingested into FHIR, search for the table name under **Tables and views**. You can see that it is of type **string**.

Example 1: Calculating a value for age

In this sample SQL query, we use a built-in SQL tool, `current_date` and `year` to extract those components. Then, we subtract them to return a patient's actual age as a column called `age`.

```
SELECT
 (year(current_date) - year(date(birthdate))) as age
FROM patient
```

Example 2: Filtering for patients who are born before `2019-01-01` and are `male`.

The SQL query shows you how to use the `CAST` function to cast the `birthDate` element as type `DATE`, and how to filter based on two criteria in the `WHERE` clause. Because the element is ingested as type **string** by default, we must `CAST` it as type `DATE`. Then you can use the < operator to compare it to a different date, `2019-01-01`. By using `AND`, you can add a second criteria to the `WHERE` clause.

```
SELECT birthdate
FROM patient
-- we convert birthdate (varchar) to date  > cast that as date too
WHERE CAST(birthdate AS DATE) < CAST('2019-01-01' AS DATE) AND gender = 'male'
```

Resource type: Location

This example shows searches for locations within the Location resource type where the city name is Attleboro.

```
SELECT *
FROM Location
WHERE address.city='ATTLEBORO'
```

```
LIMIT 10;
```

Element: Age

```
SELECT birthdate
FROM patient
-- we convert birthdate (varchar) to date  > cast that as date too
WHERE CAST(birthdate AS DATE) < CAST('2019-01-01' AS DATE) AND gender = 'male'
```

Resource type: Condition

The resource type condition stores diagnosis data related to issues that have risen to a level of concern. HealthLake's integrated medical natural language processing (NLP) generates *new* `Condition` resources based on details found in the DocumentReference resource type. When new resource are generated, HealthLake appends the tag `SYSTEM_GENERATED` to the `meta` element. This sample SQL query demonstrates how you can search the condition table and return results where the `SYSTEM_GENERATED` results have been removed.

To learn more about HealthLake's integrated natural language processing (NLP), see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake.

```
SELECT *
FROM condition
WHERE meta.tag[1] is NULL
```

You can also search within a specified string element to filter your query further. The `modifierextension` element contains details about which `DocumentReference` resource was used to generate a set of conditions. Again, you must use `json_extract` to access the nested JSON elements that are brought into Athena as a string.

This sample SQL query demonstrates how you can search for all the `Condition` that has been generated based off of a specific `DocumentReference`. Use `CAST` to set the JSON element as a string so that you can use `LIKE` to compare.

```
SELECT
    meta.tag[1].display as SystemGenerated,
    json_extract(modifierextension[4], '$.valueReference.reference') as
 DocumentReference
```

```
FROM condition
WHERE meta.tag[1].display = 'SYSTEM_GENERATED'

AND CAST(json_extract(modifierextension[4], '$.valueReference.reference') as
 VARCHAR) LIKE '%DocumentReference/67aa0278-8111-40d0-8adc-43055eb9d18d%'
```

Resource type: Observation

The resource type, Observation stores measurements and simple assertions made about a patient, device, or other subject. HealthLake's integrated natural language processing (NLP) generates *new* `Observation` resources based on details found in a `DocumentReference` resource. This sample SQL query includes `WHERE meta.tag[1] is NULL` commented out, which means that the `SYSTEM_GENERATED` results are included.

```
SELECT valueCodeableConcept.coding[1].code
FROM Observation
WHERE  valueCodeableConcept.coding[1].code = '266919005'
-- WHERE meta.tag[1] is NULL
```

This column was imported as an `struct`. Therefore, you can access elements inside it using dot notation.

Resource type: MedicationStatement

MedicationStatement is a FHIR resource type that you can use to store details about medications a patient has taken, is taking, or will take in the future. HealthLake's integrated medical natural language processing (NLP) generates new MedicationStatement resources based on documents found in the DocumentReference resource type. When new resources are generated, HealthLake appends the tag `SYSTEM_GENERATED` to the `meta` element. This sample SQL query demonstrates how to create a query that filters based off of a single patient by using their identifier and finds resources that have been added by HealthLake's integrated NLP.

```
SELECT *
FROM medicationstatement
WHERE meta.tag[1].display = 'SYSTEM_GENERATED' AND subject.reference =
 'Patient/0679b7b7-937d-488a-b48d-6315b8e7003b';
```

To learn more about HealthLake's integrated medical NLP, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake.

Additional sample SQL queries

> **Note**
>
> After February 20, 2023, HealthLake data stores do *not* use integrated natural language processing (NLP) by default. If you are interested in turning on this feature on your data store, see How do I turn on HealthLake's integrated natural language processing feature? in the Troubleshooting chapter.

This topic demonstrates how you can author SQL queries for HealthLake's integration with Athena.

Example Creating filtering criteria that are based on demographic data

Identifying the correct patient demographics is important when creating a patient cohort. This sample query demonstrates how you can use Trino dot notation and `json_extract` to filter data in your HealthLake data store.

```
SELECT
    id
    , CONCAT(name[1].family, ' ', name[1].given[1]) as name
    , (year(current_date) - year(date(birthdate))) as age
    , gender as gender
    , json_extract(extension[1], '$.valueString') as MothersMaidenName
    , json_extract(extension[2], '$.valueAddress.city') as birthPlace
    , maritalstatus.coding[1].display as maritalstatus
    , address[1].line[1] as addressline
    , address[1].city as city
    , address[1].district as district
    , address[1].state as state
    , address[1].postalcode as postalcode
    , address[1].country as country
    , json_extract(address[1].extension[1], '$.extension[0].valueDecimal') as latitude
    , json_extract(address[1].extension[1], '$.extension[1].valueDecimal') as longitude
    , telecom[1].value as telNumber
    , deceasedboolean as deceasedIndicator
    , deceaseddatetime
FROM database.patient;
```

With the Athena console, you can further sort and download the results.

Example Creating filters for a patient and their related conditions

This sample query demonstrates how you can find and sort all the related conditions for the patients found in a HealthLake data store.

```
SELECT
 patient.id as patientId
    , condition.id  as conditionId
    , CONCAT(name[1].family, ' ', name[1].given[1]) as name
    , condition.meta.tag[1].display
    , json_extract(condition.modifierextension[1], '$.valueDecimal') AS confidenceScore
    , category[1].coding[1].code as categoryCode
    , category[1].coding[1].display as categoryDescription
    , code.coding[1].code as diagnosisCode
    , code.coding[1].display as diagnosisDescription
    , onsetdatetime
    , severity.coding[1].code as severityCode
    , severity.coding[1].display as severityDescription
    , verificationstatus.coding[1].display as verificationStatus
    , clinicalstatus.coding[1].display as clinicalStatus
    , encounter.reference as encounterId
    , encounter.type as encountertype
FROM database.patient, condition
WHERE CONCAT('Patient/', patient.id) = condition.subject.reference
ORDER BY name;
```

You can use the Athena console to further sort these results or download them for further analysis.

Example Creating filters for a patients and their related observations

This sample query demonstrates how you can find and sort all the related observations for the patients found in a HealthLake data store.

```
SELECT
 patient.id as patientId
    , observation.id as observationId
    , CONCAT(name[1].family, ' ', name[1].given[1]) as name
    , meta.tag[1].display
    , json_extract(modifierextension[1], '$.valueDecimal') AS confidenceScore
    , status
    , category[1].coding[1].code as categoryCode
    , category[1].coding[1].display as categoryDescription
    , code.coding[1].code as observationCode
```

```
    , code.coding[1].display as observationDescription
    , effectivedatetime
    , CASE
  WHEN valuequantity.value IS NOT NULL THEN CONCAT(CAST(valuequantity.value AS
 VARCHAR),' ',valuequantity.unit)
       WHEN valueCodeableConcept.coding [ 1 ].code IS NOT NULL THEN
 CAST(valueCodeableConcept.coding [ 1 ].code AS VARCHAR)
       WHEN valuestring IS NOT NULL THEN CAST(valuestring AS VARCHAR)
       WHEN valueboolean IS NOT NULL THEN CAST(valueboolean AS VARCHAR)
       WHEN valueinteger IS NOT NULL THEN CAST(valueinteger AS VARCHAR)
       WHEN valueratio IS NOT NULL THEN CONCAT(CAST(valueratio.numerator.value AS
 VARCHAR),'/',CAST(valueratio.denominator.value AS VARCHAR))
       WHEN valuerange IS NOT NULL THEN CONCAT(CAST(valuerange.low.value AS
 VARCHAR),'-',CAST(valuerange.high.value AS VARCHAR))
       WHEN valueSampledData IS NOT NULL THEN CAST(valueSampledData.data AS VARCHAR)
       WHEN valueTime IS NOT NULL THEN CAST(valueTime AS VARCHAR)
       WHEN valueDateTime IS NOT NULL THEN CAST(valueDateTime AS VARCHAR)
       WHEN valuePeriod IS NOT NULL THEN valuePeriod.start
       WHEN component[1] IS NOT NULL THEN CONCAT(CAST(component[2].valuequantity.value
 AS VARCHAR),' ',CAST(component[2].valuequantity.unit AS VARCHAR),
 '/', CAST(component[1].valuequantity.value AS VARCHAR),'
 ',CAST(component[1].valuequantity.unit AS VARCHAR))
    END AS observationvalue
 , encounter.reference as encounterId
    , encounter.type as encountertype
FROM database.patient, observation
WHERE CONCAT('Patient/', patient.id) = observation.subject.reference
ORDER BY name;
```

Example Creating filtering conditions for a patient and their related procedures

Connecting procedures to patients is an important aspect of healthcare. This SQL query demonstrates how you can use the patient and procedure resource types to do that in Athena. This SQL query will return all the patients and their related procedures found in your HealthLake data store.

```
SELECT
 patient.id  as patientId
 , PROCEDURE.id as procedureId
 , CONCAT(name[1].family, ' ', name[1].given[1]) as name
 , status
 , category.coding[1].code as categoryCode
 , category.coding[1].display as categoryDescription
```

```
 , code.coding[1].code as procedureCode
 , code.coding[1].display as procedureDescription
 , performeddatetime
 , performer[1]
 , encounter.reference as encounterId
 , encounter.type as encountertype
FROM database.patient, procedure
WHERE CONCAT('Patient/', patient.id) = procedure.subject.reference
ORDER BY name;
```

Now you can use the Athena console to download the results for further analysis or sort them to better understand the results.

Example Creating filtering conditions for a patient and their related prescriptions

Seeing a current list of medications that patients are taking is important. Using Athena, you can write a SQL query that uses both the Patient and MedicationRequest resource types found in your HealthLake data store.

This SQL query joins the Patient and MedicationRequest tables imported into Athena. It also organizes the prescriptions into their individual entries by using dot notation.

```
SELECT
 patient.id  as patientId
 , medicationrequest.id  as medicationrequestid
 , CONCAT(name[1].family, ' ', name[1].given[1]) as name
 , status
 , statusreason.coding[1].code as categoryCode
 , statusreason.coding[1].display as categoryDescription
 , category[1].coding[1].code as categoryCode
 , category[1].coding[1].display as categoryDescription
 , priority
 , donotperform
 , encounter.reference as encounterId
 , encounter.type as encountertype
 , medicationcodeableconcept.coding[1].code as medicationCode
 , medicationcodeableconcept.coding[1].display as medicationDescription
 , dosageinstruction[1].text as dosage
FROM database.patient, medicationrequest
WHERE CONCAT('Patient/', patient.id ) = medicationrequest.subject.reference
ORDER BY name
```

You can use the Athena console to sort the results or download them for further analysis.

Example Seeing medications found in the MedicationStatement resource type

The example query shows you how to organize the nested JSON imported into Athena using SQL. The query uses the `meta` element to indicate when a medication has been added by HealthLake's integrated natural language processing (NLP). To learn more about HealthLake's integration with Amazon Comprehend Medical, see Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake. It also uses `json_extract` to search for data inside the array of JSON strings.

```
SELECT
 medicationcodeableconcept.coding[1].code as medicationCode
 , medicationcodeableconcept.coding[1].display as medicationDescription
 , meta.tag[1].display
 , json_extract(modifierextension[1], '$.valueDecimal') AS confidenceScore
FROM medicationstatement;
```

You can use the Athena console to download these results or sort them.

Example Filter for a specific disease type

The example shows how you can find a group of patients, aged 18 to 75, who have been diagnosed with diabetes.

```
SELECT patient.id as patientId,
 condition.id as conditionId,
 CONCAT(name [ 1 ].family, ' ', name [ 1 ].given [ 1 ]) as name,
 (year(current_date) - year(date(birthdate))) AS age,
 CASE
  WHEN condition.encounter.reference IS NOT NULL THEN condition.encounter.reference
  WHEN observation.encounter.reference IS NOT NULL THEN observation.encounter.reference
 END as encounterId,
 CASE
  WHEN condition.encounter.type IS NOT NULL THEN observation.encounter.type
  WHEN observation.encounter.type IS NOT NULL THEN observation.encounter.type
 END AS encountertype,
 condition.code.coding [ 1 ].code as diagnosisCode,
 condition.code.coding [ 1 ].display as diagnosisDescription,
 observation.category [ 1 ].coding [ 1 ].code as categoryCode,
 observation.category [ 1 ].coding [ 1 ].display as categoryDescription,
 observation.code.coding [ 1 ].code as observationCode,
 observation.code.coding [ 1 ].display as observationDescription,
 effectivedatetime AS observationDateTime,
```

```
 CASE
      WHEN valuequantity.value IS NOT NULL THEN CONCAT(CAST(valuequantity.value AS
 VARCHAR),' ',valuequantity.unit)
      WHEN valueCodeableConcept.coding [ 1 ].code IS NOT NULL THEN
 CAST(valueCodeableConcept.coding [ 1 ].code AS VARCHAR)
      WHEN valuestring IS NOT NULL THEN CAST(valuestring AS VARCHAR)
      WHEN valueboolean IS NOT NULL THEN CAST(valueboolean AS VARCHAR)
      WHEN valueinteger IS NOT NULL THEN CAST(valueinteger AS VARCHAR)
      WHEN valueratio IS NOT NULL THEN CONCAT(CAST(valueratio.numerator.value AS
 VARCHAR),'/',CAST(valueratio.denominator.value AS VARCHAR))
      WHEN valuerange IS NOT NULL THEN CONCAT(CAST(valuerange.low.value AS
 VARCHAR),'-',CAST(valuerange.high.value AS VARCHAR))
      WHEN valueSampledData IS NOT NULL THEN CAST(valueSampledData.data AS VARCHAR)
      WHEN valueTime IS NOT NULL THEN CAST(valueTime AS VARCHAR)
      WHEN valueDateTime IS NOT NULL THEN CAST(valueDateTime AS VARCHAR)
      WHEN valuePeriod IS NOT NULL THEN valuePeriod.start
      WHEN component[1] IS NOT NULL THEN CONCAT(CAST(component[2].valuequantity.value
 AS VARCHAR),' ',CAST(component[2].valuequantity.unit AS VARCHAR),
 '/', CAST(component[1].valuequantity.value AS VARCHAR),'
 ',CAST(component[1].valuequantity.unit AS VARCHAR))
    END AS observationvalue,
 CASE
  WHEN condition.meta.tag [ 1 ].display = 'SYSTEM GENERATED' THEN 'YES'
  WHEN condition.meta.tag [ 1 ].display IS NULL THEN 'NO'
  WHEN observation.meta.tag [ 1 ].display = 'SYSTEM GENERATED' THEN 'YES'
  WHEN observation.meta.tag [ 1 ].display IS NULL THEN 'NO'
   END AS IsSystemGenerated,
  CAST(
    json_extract(
      condition.modifierextension [ 1 ],
      '$.valueDecimal'
    ) AS int
  ) AS confidenceScore
FROM database.patient,
 database.condition,
 database.observation
WHERE CONCAT('Patient/', patient.id) = condition.subject.reference
 AND CONCAT('Patient/', patient.id) = observation.subject.reference
   AND (year(current_date) - year(date(birthdate))) >= 18
   AND (year(current_date) - year(date(birthdate))) <= 75
   AND condition.code.coding [ 1 ].display like ('%diabetes%');
```

Now you can use the Athena console to sort the results or download them for further analysis.

AWS HealthLake and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and AWS 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 AWS HealthLake

Important notice

AWS 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 AWS 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 a HealthLake data store.

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 requirementsFor 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.

Topics

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, ReadResource, SearchAll, SearchWithGet, SearchWithPost, UpdateResource.
- 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 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, data store type
User Errors	The number of requests that failed due to user error. Units: Count Valid Statistics: Sum, Average

Metrics	Description
	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 (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**.

Integrating SMART on FHIR with AWS HealthLake

A Substitutable Medical Applications and Reusable Technologies (SMART) on FHIR enabled HealthLake data store allows SMART on FHIR compliant applications to access data stored in a HealthLake data store. HealthLake data is accessed by authenticating and authorizing requests using a third-party authorization server, and by setting up additional resources in AWS.

To use the SMART on FHIR with your HealthLake data store you must provide the following in your CreateFHIRDatastore API request.

- Set the AuthorizationStrategy equal to `SMART_ON_FHIR_V1`.
- Set the IdpLambdaArn equal to the ARN of the AWS Lambda you created to manage token decoding with your authorization server.
- Define the Metadata elements specified in your authorization server. These metadata elements are returned in the Discovery Document. To learn more, see Fetching a SMART on FHIR enabled HealthLake data store's Discovery Document.
- *Optional*: Enable FineGrainedAuthorizationEnabled if you've set up fine grained authorization on your authorization server.

You can make a SMART on FHIR enabled data store using the AWS Command Line Interface (AWS CLI) or via one of the AWS supported SDKs. Creating a SMART on FHIR enabled HealthLake data store is not supported using the HealthLake console. To learn more, see Create a SMART on FHIR enabled data store.

To prescribe these parameters in the request, you need to set up resources in other AWS services (AWS Secrets Manager and AWS Lambda), create new IAM service roles, and set up a SMART on FHIR compliant authorization server. Use the section Setting up the required resources to implement a SMART on FHIR compliant data store to learn more about setting up the required resources and to see a high-level overview of how a SMART on FHIR Application interacts with HealthLake.

This means that instead of managing user credentials via AWS Identity and Access Management you are doing so using a SMART on FHIR compliant authorization server.

HealthLake supports SMART on FHIR 1.0. To learn more about this framework, see SMART Application Launch Framework Implementation Guide Release 1.0.

To authorize and authenticate requests for data store using SMART on FHIR, HealthLake supports using:

- **OpenID (AuthN) Integration**: Used to authenticate that person or client application is who (or what) they claim to be.
- **OAuth 2.0 (AuthZ) Integration**: Used to authorize what FHIR resources in your HealthLake data store an authenticated request can read or write data too. This is defined by the scopes set up in your authorization server

Contents

- An example request from client application containing a JWT in the authorization header and how Lambda should decode that response
- Setting up resources needed to implement a SMART on FHIR compliant data store
  - How a client application launches and requests data from a SMART on FHIR enable HealthLake data store

Authentication requirements for SMART on FHIR

To access FHIR resources in a SMART on FHIR HealthLake data store, a client application must be authorized by an OAuth 2.0-compliant authorization server and present an OAuth Bearer token as part of a FHIR REST API request. To find the authorization server's endpoint use the HealthLake SMART on FHIR Discovery Document via a Well-Known Uniform Resource Identifier. To learn more about this process, see Fetching a SMART on FHIR enabled HealthLake data store's Discovery Document.

When you create a SMART on FHIR HealthLake data store, you must define the authorization server's end point and the token endpoint in the `metadata` element of the CreateFHIRDatastore request. To learn more defining the `metadata` element, see Creating a SMART on FHIR enabled HealthLake data store.

Using the authorization server endpoints, the client application will authenticate a user with the authorization service. Once authorized and authenticated, a JSON Web Token (JWT) is generated by the authorization service and passed to the client application. This token contains FHIR resource scopes that the client application is allowed to use, which in turn restricts what data the user is able to access. Optionally, if the launch scope was provided then the response will contain those details. To learn more about the SMART on FHIR scopes supported by HealthLake, see Supported SMART on FHIR OAuth scopes by HealthLake.

Using the JWT granted by the authorization server, a client application makes FHIR REST API calls to a SMART on FHIR enabled HealthLake data store. To validate and decode the JWT, you need to create a Lambda function. HealthLake invokes this Lambda function on your behalf when a FHIR REST API request is received. To see an example starter Lambda function, see Using AWS Lambda for token validation with a SMART on FHIR enabled HealthLake data store.

Authorization server elements required to create a SMART on FHIR enabled HealthLake data store

In the CreateFHIRDatastore request, you need to provide the authorization endpoint and the token endpoint as part of the `metadata` element in the `IdentityProviderConfiguration` object. Both the authorization endpoint and token endpoint are required. To see example of how this is specifed in CreateFHIRDatastore request, see Creating a SMART on FHIR enabled HealthLake data store.

Required claims to complete a FHIR REST API request on a SMART on FHIR enabled HealthLake data store

Your AWS Lambda function must contain the following claims for it to be a valid FHIR REST API request on a SMART on FHIR enabled HealthLake data store.

- `nbf`: (Not Before) Claim — The "nbf" (not before) claim identifies the time before which the JWT MUST NOT be accepted for processing. The processing of the "nbf" claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the "nbf" claim. The sample Lambda function we provide converts `iat` from the server response into `nbf`.
- `exp`: (Expiration Time) Claim — The "exp" (expiration time) claim identifies the expiration time on or after which the JWT must not be accepted for processing.
- `isAuthorized`: A boolean set to `True`. Indicates that request has been authorized on the authorization server.
- `aud`: (Audience) Claim — The "aud" (audience) claim identifies the recipients that the JWT is intended for. This must be a SMART on FHIR enabled HealthLake data store endpoint.
- `scope`: This must be at least one FHIR resource related scope. This scope is defined on your authorization server. To learn more about FHIR resource related scopes accepted by HealthLake, see HealthLake data store FHIR resource specific scopes.

Supported SMART on FHIR OAuth scopes by HealthLake

HealthLake uses OAuth 2.0 as an authorization protocol. Using this protocol on your authorization server allows you to define what FHIR resources in your HealthLake data store a client application can have read and/or write access too.

The SMART on FHIR framework defines a set of scopes that can be requested from the authorization server. To view the scope definitions in the SMART on FHIR framework, see SMART on FHIR Scopes in the *HL7 FHIR Resource Guide*.

For example, a client application that is only designed to allow patients to view their lab results or view their contact details should only be *authorized* to request (via FHIR REST request) `read` scopes. To define these as scope you would provide a string like the following `patient/Observation.read`. This would allow the client application to request access to the `Observation` resource type in a read-only manner on the `Patient` resource type.

Standalone launch scope

HealthLake supports the standalone launch mode scope `launch/patient`.

In standalone launch mode a client application requests access to patient's clinical data because the user and patient are not known to the client application. Thus, the client application's authorization request explicitly requests the patient scope be returned. After successful authentication, the authorization server issues a access token containing the requested launch patient scope. The needed patient context is provided alongside the access token in the authorization server's response.

Supported launch mode scopes

Scope	Description
`launch/patient`	A parameter in a OAuth 2.0 authorization request requesting that patient data be returned in the authorization response.

HealthLake data store FHIR resource specific scopes

HealthLake defines three levels of scopes.

- Patient-specific scopes grant access to specific data about a single patient. Which patient is specified in the launch context.
- User-level scopes grant access to specific data that a user can access.
- System-level scopes grant read/write access to all FHIR resource found in the HealthLake data store.

The following table shows the syntax for constructing FHIR resource related scopes that are supported by HealthLake. The general format is the following:

```
( 'patient' | 'user' | 'system' ) '/' ( fhir-resource | '*' ) '.' ( 'read' | 'write' |
 '*' )
```

Supported authorization scopes on HealthLake data stores

Scope syntax	Example scope	Result
`patient/(fhir-resource | '*').('read' | 'write' | '*')`	`patient/AllergyIntolerance.*`	A client application would have read/write access to allergies.
`user/(fhir-resource | '*').('read' | 'write' | '*')`	`user/Observation.read`	A client application would have read access to all recorded observations.
`system/('read' | 'write' | *)`	`system/*.*`	A client application would have read/write access to all data.

Using AWS Lambda for token validation with a SMART on FHIR enabled HealthLake data store

When you create a SMART on FHIR enabled HealthLake data store, you need to provide the ARN of the AWS Lambda function in the `CreateFHIRDatastore` request. The Lambda function's ARN is specified in `IdentityProviderConfiguration` object using the `IdpLambdaArn` parameter.

You must create the Lambda function prior to creating your SMART on FHIR enabled HealthLake data store. Once you create the data store, the Lambda ARN cannot be changed. To see the Lambda ARN you specified when the data store was created use the `DescribeFHIRDatastore` API operation.

For a FHIR REST request to succeed on a SMART on FHIR enabled HealthLake data store your Lambda function needs to do the following:

- The Lambda function must return a response in less than 1 second to HealthLake data store endpoint.
- Decode the access token provided in the authorization header of the REST API request sent by the client application.
- Assign an IAM service role that has sufficient permissions to carry out the FHIR REST API request.
- The following claims are required to complete a FHIR REST API request. To learn more, see Required claims.
  - `nbf`
  - `exp`
  - `isAuthorized`
  - `aud`
  - `scope`

When working with Lambda, you need to create an execution role and a resource-based policy in addition to your Lambda function. A Lambda's function's execution role is an IAM role that grants the function permission to access AWS services and resources needed at run time. The resource-based policy you provide must allow HealthLake to invoke your function on your behalf.

The sections in this topic describe an example request from a client application and decoded response, the steps needed to create an AWS Lambda function, and how to create a resource-based policy that HealthLake can assume.

- Part 1: Creating a Lambda function
- Part 2: Creating a HealthLake service role used by the AWS Lambda function
- Part 3: Updating the Lambda function's execution role
- Part 4: Adding a resource policy to your Lambda function
- Part 5: Provisioning concurrency for your Lambda function

Creating an AWS Lambda function

The Lambda function created in this topic is triggered when HealthLake receives a requests to a SMART on FHIR enabled HealthLake data store. The request from the client application contains a REST API call, and authorization header containing an access token.

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Authorization: Bearer i8hweunweunweofiwweoijewiwe
```

The example Lambda function in this topic uses AWS Secrets Manager to obscure credentials related to the authorization server. We strongly recommend not providing authorization server login details directly in a Lambda function.

Example validating a FHIR REST request containing an authorization bearer token

The example Lambda function shows you how to validate an FHIR REST request sent to a SMART on FHIR enabled HealthLake data store. To see step-by-steps directions on how to implement this Lambda function, see Creating a Lambda function using the AWS Management Console.

If the FHIR REST API request does not contain a valid data store endpoint, access token, and REST operation the Lambda function will fail. To learn more about the required authorization server elements, see Required claims.

```
import base64
import boto3
import logging
import json
import os
from urllib import request, parse

logger = logging.getLogger()
logger.setLevel(logging.INFO)

## Uses Secrets manager to gain access to the access key ID and secret access key for
 the authorization server
client = boto3.client('secretsmanager', region_name="region-of-datastore")
response = client.get_secret_value(SecretId='name-specified-by-customer-in-
secretsmanager')
secret = json.loads(response['SecretString'])
client_id = secret['client_id']
client_secret = secret['client_secret']
```

```
unencoded_auth = f'{client_id}:{client_secret}'
headers = {
  'Authorization': f'Basic {base64.b64encode(unencoded_auth.encode()).decode()}',
  'Content-Type': 'application/x-www-form-urlencoded'
}

auth_endpoint = os.environ['auth-server-base-url'] # Base URL of the Authorization
 server
user_role_arn = os.environ['iam-role-arn'] # The IAM role client application will use
 to complete the HTTP request on the datastore

def lambda_handler(event, context):
    if 'datastoreEndpoint' not in event or 'operationName' not in event or
 'bearerToken' not in event:
    return {}

    datastore_endpoint = event['datastoreEndpoint']
    operation_name = event['operationName']
    bearer_token = event['bearerToken']
    logger.info('Datastore Endpoint [{}], Operation Name:
 [{}]'.format(datastore_endpoint, operation_name))

    ## To validate the token
    auth_response = auth_with_provider(bearer_token)
    logger.info('Auth response: [{}]'.format(auth_response))
    auth_payload = json.loads(auth_response)
    ## Required parameters needed to be sent to the datastore endpoint for the HTTP
 request to go through
    auth_payload["isAuthorized"] = bool(auth_payload["active"])
    auth_payload["nbf"] = auth_payload["iat"]
    return {"authPayload": auth_payload, "iamRoleARN": user_role_arn}

## access the server
def auth_with_provider(token):
    data = {'token': token, 'token_type_hint': 'access_token'}
    req = request.Request(url=auth_endpoint + '/v1/introspect',
 data=parse.urlencode(data).encode(), headers=headers)
    with request.urlopen(req) as resp:
    return resp.read().decode()
```

Creating a Lambda function using the AWS Management Console

This procedure assumes you already created the service role that you want HealthLake to assume when handling a FHIR REST API request on a SMART on FHIR enabled HealthLake data store. If you have not created the service role, you can still create the Lambda function. You will need to add the ARN of service role before the Lambda function will work. To learn more about creating a service role and specifying it in the Lambda function see, Creating a HealthLake service role for use in the AWS Lambda function used to decode a JWT

To create a Lambda function (AWS Management Console)

1. Open the Functions page of the Lambda console.
2. Choose **Create function**.
3. Select **Author from scratch**.
4. Under **Basic information** enter a **Function name**. Under **Runtime** choose a python based runtime.
5. For **Execution role**, choose **Create a new role with basic Lambda permissions**.

   Lambda creates an execution role that grants the function permission to upload logs to Amazon CloudWatch. The Lambda function assumes the execution role when you invoke your function, and uses the execution role to create credentials for the AWS SDK.
6. Choose the **Code** tab, and add the sample Lambda function.

   If you've not yet created the service role for the Lambda function to use you'll need to create it before the sample Lambda function will work. To learn more about creating a service role for the Lambda function, see Creating a HealthLake service role for use in the AWS Lambda function used to decode a JWT.

```
import base64
import boto3
import logging
import json
import os
from urllib import request, parse

logger = logging.getLogger()
logger.setLevel(logging.INFO)
```

```
## Uses Secrets manager to gain access to the access key ID and secret access key
 for the authorization server
client = boto3.client('secretsmanager', region_name="region-of-datastore")
response = client.get_secret_value(SecretId='name-specified-by-customer-in-
secretsmanager')
secret = json.loads(response['SecretString'])
client_id = secret['client_id']
client_secret = secret['client_secret']


unencoded_auth = f'{client_id}:{client_secret}'
headers = {
  'Authorization': f'Basic {base64.b64encode(unencoded_auth.encode()).decode()}',
  'Content-Type': 'application/x-www-form-urlencoded'
}

auth_endpoint = os.environ['auth-server-base-url'] # Base URL of the Authorization
 server
user_role_arn = os.environ['iam-role-arn'] # The IAM role client application will
 use to complete the HTTP request on the datastore

def lambda_handler(event, context):
    if 'datastoreEndpoint' not in event or 'operationName' not in event or
 'bearerToken' not in event:
    return {}

    datastore_endpoint = event['datastoreEndpoint']
    operation_name = event['operationName']
    bearer_token = event['bearerToken']
    logger.info('Datastore Endpoint [{}], Operation Name:
 [{}]'.format(datastore_endpoint, operation_name))

    ## To validate the token
    auth_response = auth_with_provider(bearer_token)
    logger.info('Auth response: [{}]'.format(auth_response))
    auth_payload = json.loads(auth_response)
    ## Required parameters needed to be sent to the datastore endpoint for the HTTP
 request to go through
    auth_payload["isAuthorized"] = bool(auth_payload["active"])
    auth_payload["nbf"] = auth_payload["iat"]
    return {"authPayload": auth_payload, "iamRoleARN": user_role_arn}

## Access the server
def auth_with_provider(token):
```

```
    data = {'token': token, 'token_type_hint': 'access_token'}
    req = request.Request(url=auth_endpoint + '/v1/introspect',
 data=parse.urlencode(data).encode(), headers=headers)
    with request.urlopen(req) as resp:
    return resp.read().decode()
```

Modifying a Lambda function's execution role

After creating the Lambda function, you need to update the execution role to include the necessary permissions to call Secrets Manager. In Secrets Manager, each secret you create has an ARN. To apply the least privilege, the execution role should only have access to the resources needed for the Lambda function to execute.

You can modify a Lambda function's execution role by searching for it in the IAM console or by choosing **Configuration** in the Lambda console. To learn more about managing your Lambda functions execution role, see Lambda execution role.

Example Lambda function execution role that grants access to `GetSecretValue`

Adding the IAM action `GetSecretValue` to execution role grants the necessary permission for the sample Lambda function to work.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "arn:aws:secretsmanager:your-region:your-aws-account-
id:secret:secret-name-DKodTA"
    }
  ]
}
```

At this point you've created a Lambda function that can be used to validate the access token provided as part of the FHIR REST request sent to your SMART on FHIR enabled HealthLake data store.

Creating a HealthLake service role for use in the AWS Lambda function used to decode a JWT

Persona: IAM Administrator

A user who can add or remove IAM policies, and create new IAM identities.

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*.

After the JSON Web Token (JWT) is decoded the authorization Lambda needs to also return an IAM role ARN. This role must have the necessary permissions to carry out the REST API request or it will fail due to insufficient permissions.

When setting up a custom policy using IAM it is best to grant the minimum permissions required. To learn more, see Apply least-privilege permissions in the *IAM User Guide*.

Creating a HealthLake service role to designate in the authorization Lambda function requires two steps.

- First, you need to create IAM policy. The policy must specify access to the FHIR resources that you have provided scopes for in the authorization server.
- Second, you need to create the service role. When you create the role you designate a trust relationship and attach the policy you created in step one. The trust relationship designates HealthLake as the service principal. You need to specify a HealthLake data store ARN and a AWS account ID in this step.

Creating a new IAM policy

The scopes you define in your authorization server determine what FHIR resources an authenticated user has access to in a HealthLake data store.

The IAM policy you create can be tailored to match the scopes you've defined.

The following actions in the `Action` element of an IAM policy statement can be defined. For each `Action` in the table you can define a `Resource types`. In HealthLake a data store is the only supported resource type that can be defined in the `Resource` element of an IAM permission policy statement.

Individual FHIR resources are not a resource that you can define as an element in a IAM permission policy.

Actions defined by HealthLake

Actions	Description	Acces level	Resource type (Required)
CreateResource	Grants permission to a create resource	Write	Datastore ARN: arn:aws:healthlake:**your-region** :**111122223333** :datastore/fhir/**your-datastore-id**
DeleteResource	Grants permission to delete resource	Write	Datastore ARN: arn:aws:healthlake:**your-region** :**111122223333** :datastore/fhir/**your-datastore-id**
ReadResource	Grants permission to read resource	Read	Datastore ARN: arn:aws:healthlake:**your-region** :**111122223333** :datastore/fhir/**your-datastore-id**
SearchWithGet	Grants permission to search resources with GET method	Read	Datastore ARN: arn:aws:healthlake:**your-region** :**111122223333** :datastore/fhir/**your-datastore-id**
SearchWithPost	Grants permission to search resources with POST method	Read	Datastore ARN: arn:aws:healthlake:**your-region** :**111122223333** :datastore/fhir/**your-datastore-id**
StartFHIRExportJobWithPost	Grants permission to begin a FHIR Export job with GET	Write	Datastore ARN: arn:aws:healthlake:**your-region** :**111122223333** :datastore/fhir/**your-datastore-id**

Actions	Description	Access level	Resource type (Required)
UpdateResource	Grants permission to update resource	Write	Datastore ARN: arn:aws:healthlake:**your-region** :**111122223333** :datastore/fhir/**your-datastore-id**

To get started, you can use `AmazonHealthLakeFullAccess`. This policy would grant read, write, search, and export on all FHIR resources found in a data store. To grant read-only permissions on a data store use `AmazonHealthLakeReadOnlyAccess`.

To learn more about creating a custom policy using the AWS Management Console, AWS CLI, or IAM SDKs, see Creating IAM policies in the *IAM User Guide*.

Creating a service role for HealthLake (IAM console)

Use this procedure to create a service role. When you create a service you will also need to designate an IAM policy.

To create the service role for HealthLake (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**.
3. Then, choose **Create role**.
4. On the **Select trust entity** page, choose **Custom trust policy**.
5. Next, under **Custom trust policy** update the sample policy as follows. Replace **your-account-id** with your account number, and add the ARN of the data store you want to use in your import or export jobs.

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

```
        },
        "Condition": {
          "StringEquals": {
            "aws:SourceAccount": "your-account-id"
            },
            "ArnEquals": {
              "aws:SourceArn": "arn:aws:healthlake:your-region:your-account-
id:datastore/fhir/your-datastore-id"
            }
          }
        }
  ]
}
```

6. Then, choose **Next**.
7. On the **Add permissions** page, choose the policy that you want the HealthLake service to assume. To find your policy, search for it under **Permissions policies**.
8. Then, choose **Attach policy**.
9. Then on the **Name, review, and create** page under **Role name** enter a name.
10. (Optional)Then under **Description**, add a short description for your role.
11. If possible, enter a role name or role name suffix to help you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **`PRODROLE`** and **`prodrole`**. Because various entities might reference the role, you cannot edit the name of the role after it has been created.
12. Review the role details, and then choose **Create role**.

To learn how to specify the role ARN in the sample Lambda function, see Creating an AWS Lambda function.

Lambda execution role

A Lambda function's execution role is an IAM role that grants the function permission to access AWS services and resources. This page provides information on how to create, view, and manage a Lambda function's execution role.

By default, Lambda creates an execution role with minimal permissions when you create a new Lambda function using the AWS Management Console. To manage the permissions granted in the execution role, see Creating an execution role in the IAM console in the *Lambda Developer Guide*.

The sample Lambda function provided in this topic uses Secrets Manager to obscure the authorization server's credentials.

As with any IAM role you create it is important to follow the least privilege best practice. During the development phrase, you might sometimes grant permissions beyond what is required. Before publishing your function in the production environment, as a best practice, adjust the policy to include only the required permissions. For more information, see Apply least-privelege in the *IAM User Guide*.

Allow HealthLake to trigger your Lambda function

So HealthLake can invoke the Lambda function on your behalf, you must do following:

- You need to set `IdpLambdaArn` equal to the ARN of the Lambda function you want HealthLake to invoke in the `CreateFHIRDatastore` request.
- You need a resource-based policy allowing HealthLake to invoke the Lambda function on your behalf.

When HealthLake receives a FHIR REST API request on a SMART on FHIR enabled HealthLake data store, it needs permissions to invoke the Lambda function specified at data store creation on your behalf. To grant HealthLake access, you'll use a resource-based policy. To learn more about creating a resource-based policy for a Lambda function, see Allowing an AWS service to call a Lambda function in the *AWS Lambda Developer Guide*.

Provisioning concurrency for your Lambda function

> **Important**
>
> HealthLake requires that the maximum run time for your Lambda function be less than one second (1000 milliseconds).
> If you Lambda function exceeds the run time limit you get a TimeOut exception.

To avoid getting this exception, we recommend configuring provisioned concurrency. By allocating provisioned concurrency before an increase in invocations, you can ensure that all requests are

served by initialized instances with low latency. To learn more about configuring provisioned concurrency, see Configuring provisioned concurrency in the *Lambda Developer Guide*

To see the average run time for your Lambda function currently use the **Monitoring** page for your Lambda function on the Lambda console. By default, the Lambda console provides a **Duration** graph which shows you the average, minimum, and maximum amount of time your function code spends processing an event. To learn more about monitoring Lambda functions, see Monitoring functions in the Lambda console in the *Lambda Developer Guide*.

If you have already provisioned concurrency for your Lambda function and want to monitor it, see Monitoring concurrency in the *Lambda Developer Guide*.

Creating a SMART on FHIR enabled HealthLake data store

To use the SMART on FHIR framework with HealthLake, create a HealthLake data store with the `IdentityProviderConfiguration` parameter specified in your CreateFHIRDatastore request. In the `IdentityProviderConfiguration` parameter you specify the following information:

- Set the AuthorizationStrategy equal to `SMART_ON_FHIR_V1`.
- Set the IdpLambdaArn equal to the ARN of the AWS Lambda you created to manage token decoding with your authorization server.
- Define the Metadata elements specified in the authorization server as a JSON block. These metadata elements are returned in the Discovery Document.
- *Optional*: Enable FineGrainedAuthorizationEnabled. Specify `True` to use the Fine grained authorization provided by HealthLake

You can make a SMART on FHIR enabled data store using the AWS Command Line Interface (AWS CLI) or via one of the AWS supported SDKs. Creating a SMART on FHIR enabled HealthLake data store is not supported using the HealthLake console.

Using the AWS CLI to create a SMART on FHIR enabled HealthLake data store

You can use the following code example to create SMART on FHIR enabled HealthLake data store using the AWS CLI. When creating a SMART on FHIR enabled HealthLake data store you must specify the `identity-provider-configuration` parameter.

In the `identity-provider-configuration` parameter you can *optionally* enable fine-grained authorization by setting `FineGrainedAuthorizationEnabled` equal to `True`. To learn more about Fine grained authorization, see Using fine-grained authorization with a SMART on FHIR enabled HealthLake data store. The example below contains a special character \ to indicate line breaks or as an escape character. This is for clarity.

```
aws healthlake create-fhir-datastore \
  --region us-east-1 \
  --datastore-name "your-data-store-name" \
  --datastore-type-version R4 \
  --preload-data-config PreloadDataType="SYNTHEA" \
  --sse-configuration '{ "KmsEncryptionConfig": { \
    "CmkType": "customer-managed-kms-key1",
    "KmsKeyId": "arn:aws:kms:us-east-1:your-account-id:key/your-key-id" } }' \
  --identity-provider-configuration  \
      '{"AuthorizationStrategy": "SMART_ON_FHIR_V1", \
      "FineGrainedAuthorizationEnabled": boolean-false-by-default, \
      "IdpLambdaArn": "arn:aws:lambda:your-region:your-account-id:function:your-lambda-
name" \
      "Metadata": "{\"issuer\":\"https://ehr.example.com\",\"jwks_uri\":\"https://
ehr.example.com/.well-known/jwks.json\",\"authorization_endpoint\":\"https://
ehr.example.com/auth/authorize\",\"token_endpoint\":\"https://ehr.token.com/auth/
token\",\"token_endpoint_auth_methods_supported\":[\"client_secret_basic\",\"foo\"],
\"grant_types_supported\":[\"client_credential\",\"foo\"],\"registration_endpoint\":
\"https://ehr.example.com/auth/register\",\"scopes_supported\":[\"openId\",\"profile\",
\"launch\"],\"response_types_supported\":[\"code\"],\"management_endpoint\":\"https://
ehr.example.com/user/manage\",\"introspection_endpoint\":\"https://ehr.example.com/
user/introspect\",\"revocation_endpoint\":\"https://ehr.example.com/user/revoke\",
\"code_challenge_methods_supported\":[\"S256\"],\"capabilities\":[\"launch-ehr\",\"sso-
openid-connect\",\"client-public\"]}"}'
```

When successful you get the following JSON response:

```
{
  "DatastoreArn": "arn:aws:healthlake:your-region:111122223333:datastore/fhir/your-
datastore-id",
  "DatastoreEndpoint": "https://healthlake.your-region.amazonaws.com/datastore/your-
datastore-id/r4/",
  "DatastoreId": "your-data-store-id",
  "DatastoreStatus": "data-store-creation-status"
}
```

Using fine-grained authorization with a SMART on FHIR enabled HealthLake data store

Scopes alone do not provide you with the necessary specificity about what data a requester is authorized to access in a data store. Using fine-grained authorization enables a higher level of specificity when granting access to a SMART on FHIR enabled HealthLake data store. To use fine-grained authorization, set `FineGrainedAuthorizationEnabled` equal to `True` in the `IdentityProviderConfiguration` parameter of your CreateFHIRDatastore request.

If you enabled fine-grained authorization, your authorization server returns a `fhirUser` scope in the `id_token` along with the access token. This permits information about the User to be retrieved by client application. The client application should treat the `fhirUser` claim as the URI of a FHIR resource representing the current user. This can be `Patient`, `Practitioner`, or `RelatedPerson`. The authorization server's response also includes a `user/` scope that defines what data the user can access. This uses the syntax defined for scopes related to FHIR resource specific scopes:

```
user/(fhir-resource | '*').('read' | 'write' | '*')
```

The following are examples of how fine-grained authorization can be used to further specify data access related FHIR resource types.

- When `fhirUser` is a `Practitioner`, fine-grained authorization determines the collection of patients that the user can access. Access to `fhirUser` is allowed for only those patients where the Patient has reference to the `fhirUser` as a General Practitioner.

  ```
  Patient.generalPractitioner : [{Reference(Practitioner)}]
  ```

- When `fhirUser` is a `Patient` or `RelatedPerson` and the patient referenced in the request is different from the `fhirUser`, fine-grained authorization determines access to `fhirUser` for the requested patient. Access is allowed when there is a relationship specified in requested `Patient` resource.

  ```
  Patient.link.other : {Reference(Patient|RelatedPerson)}
  ```

Fetching a SMART on FHIR enabled HealthLake data store's Discovery Document

For a client application to make a successful FHIR REST request, it needs to gather the authorization requirements defined in the HealthLake data store. No authorization (bearer token) is required for this request to succeed.

To do so, make a `GET` request and append `/.well-known/smart-configuration` to the endpoint of the data store

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/.well-known/smart-configuration
```

This returns the HealthLake data store's Discovery Document as a JSON blob. In it, you will find the `authorization_endpoint` and the `token_endpoint` along with the specifications and capabilities defined in the HealthLake data store.

```
{
  "authorization_endpoint": "https://oidc.example.com/authorize",
  "token_endpoint": "https://oidc.example.com/oauth/token",
  "capabilities": [
    "launch-ehr",
    "client-public"
  ]
}
```

URLs needed for launching a client application successfully

- **Authorization endpoint**: The URL needed to authorize a client application or user.
- **Token endpoint**: The endpoint of the authorization server that the client application uses to communicate with it.

Making a FHIR REST API request on a SMART enabled HealthLake data store

An example request from client application containing a JWT in the authorization header and how Lambda should decode that response

After the client application request has been authorized and authenticated the client application must receives a bearer token from the authorization server. Use the bearer token in the authorization header when sending a FHIR REST API request on a SMART on FHIR enabled HealthLake data store.

Sample FHIR REST API request to SMART on FHIR enabled a HealthLake data store

An example GET request on a SMART on FHIR enabled HealthLake data store

```
GET https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/
Patient/[ID]
Authorization: Bearer auth-server-provided-bearer-token
```

Because a bearer token was found in the authorization header and no AWS IAM identity was detected HealthLake invokes the Lambda function specified when the SMART on FHIR enabled HealthLake data store was created. When the token is successfully decoded by your Lambda function here is an example response which sent to HealthLake.

```
{
  "authPayload": {
    "iss": "https://authorization-server-endpoint/oauth2/token", # The issuer
 identifier of the authorization server
    "aud": "https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/
r4/", # Required, data store endpoint
    "iat": 1677115637,  # Identifies the time at which the token was issued
    "nbf": 1677115637,  # Required, the earliest time the JWT would be valid
    "exp": 1997877061,  # Required, the time at which the JWT is no longer valid
    "isAuthorized": "true",  # Required, boolean indicating the request has been
 authorized
    "uid": "100101",  # Unique identifier returned by the auth server
    "scope": "system/*.*" # Required, the scope of the request
  },
  "iamRoleARN": "iam-role-arn" #Required, IAM role to complete the request
```

```
}
```

Setting up resources needed to implement a SMART on FHIR compliant data store

This topic describes the resources that you need to provision in your AWS account outside HealthLake, creating a SMART on FHIR enabled HealthLake data store, and how a SMART on FHIR client application would interact with an authorization server and a HealthLake data store.

The steps in this workflow define the basic steps for how SMART on FHIR requests are handled, and what resources are needed for them to succeed.

In a SMART on FHIR request process, three applications work together:

- **The End-User**: Generally, a patient or clinician using a third-party SMART on FHIR Application to access data in a HealthLake data store.
- **The SMART on FHIR Application (referred to as the client application)**: An application that wants to access data found in HealthLake data store.
- **The Authorization Server**: An OpenID Connect compliant server that is able to authenticate users and issue Access Tokens.
- **The HealthLake data store**: A SMART on FHIR enabled HealthLake data store that uses a Lambda function to respond to FHIR REST requests which provide a bearer token.

For these application to work together you need to create the following resources.

We recommend creating the SMART on FHIR enabled HealthLake data store after you have set up the authorization server, defined the neccessary scopes on it, and created a AWS Lambda function to handle token introspection.

1. Setting up an authorization server endpoint — Authorization server

To use the SMART on FHIR framework you need to set up an third-party authorization server that can validate FHIR REST requests made on a data store. To learn more about setting up an authorization server endpoint that will work with HealthLake, see Authentication requirements for SMART on FHIR.

2. Define scopes to control who can access what data in your HealthLake data store on your authorization server — Authorization server

The SMART on FHIR framework uses OAuth scopes to determine what FHIR resources an authenticated request has access to and to what extent. Defining scopes are a way to design for least-privilege. To learn more about scopes defined by the SMART on FHIR framework and supported by HealthLake see, Supported SMART on FHIR OAuth scopes by HealthLake.

3. Setup a AWS Lambda function capable of performing token introspection —your AWS account

A FHIR REST request sent by the client application on a SMART on FHIR enabled data store will contain a JSON Web Token (JWT). To learn more about setting up a Lambda function capable decoding and validating it, see Decoding a JWT.

4. Create a SMART on FHIR enabled HealthLake data store — your AWS account

To create a SMART on FHIR HealthLake data store you need to provide an `IdentityProviderConfiguration`. To learn more the required `IdentityProviderConfiguration` parameters in a CreateFHIRDatastore request, see Creating a SMART on FHIR enabled HealthLake data store.

How a client application launches and requests data from a SMART on FHIR enable HealthLake data store

This section explain how a client application launches with in the SMART on FHIR context, and is able to make a sucessful FHIR REST request on an HealthLake data store.

1. Client application makes a GET request to Well-Known Uniform Resource Identifier

A SMART enabled client application needs to make a GET request to find the authorization endpoints of your HealthLake data store. This is done via a Well-Known Uniform Resource Identifier (URI) request. To learn more about this, see Fetching a SMART on FHIR enabled HealthLake data store's Discovery Document.

2. Requesting access and Scopes

The client application uses the authorization endpoint of the authorization server, so that the user can login. This process authenticates the user. Scopes are used to define what FHIR resources in your HealthLake data store a client application can access. To learn more about defining scopes, see Supported SMART on FHIR OAuth scopes by HealthLake.

3. Access tokens

Now that the user has been authenticated, a client application receives a JWT access token from the authorization server. This token is provided when the client application sends a FHIR REST request to HealthLake. To learn more about how the JWT is decoded using a Lambda function, see Performing token validation.

4. Making a FHIR REST Request on SMART on FHIR enabled HealthLake data store

Now, the client application can send a FHIR REST request to a HealthLake data store endpoint using the access token provided by the authorization server. To see an example FHIR REST request, see Making a FHIR REST API request on a SMART enabled HealthLake data store.

5. Validating the JWT access token

To validate the access token sent in the FHIR REST request, use a Lambda function. To see how to create a Lambda function that can perform token introspection, see Creating an AWS Lambda function.

Using automated resource generation based on natural language processing (NLP) of the FHIR DocumentReference resource type in AWS HealthLake

> **Note**
>
> After February 20, 2023, HealthLake data stores do *not* use integrated natural language processing (NLP) by default. If you are interested in turning on this feature on your data store, see How do I turn on HealthLake's integrated natural language processing feature? in the Troubleshooting chapter.
> If you have turned on Amazon Comprehend Medical's integrated NLP, then when you create or update `DocumentReference` resources, you will incur charges in your AWS account. For more details, see AWS HealthLake pricing.
> Amazon Comprehend Medical isn't available in Asia Pacific (Mumbai). HealthLake data stores created in the Asia Pacific (Mumbai) region do not support integrated natural language processing (NLP).

HealthLake automatically provides you with integrated natural language processing (NLP) using Amazon Comprehend Medical for unstructured data processing for data stored in the `DocumentReference` resource type. To do this, HealthLake calls the Amazon Comprehend Medical `DetectEntities-V2`, `InferICD10-CM`, and `InferRxNorm` API operations. The results are automatically appended to the `DocumentReference` resource as an extension. When the Amazon Comprehend Medical API operations detect traits that are `SIGN`, `SYMPTOM`, and `DIAGNOSIS`, a `Linkage` resource type is generated automatically. New condition and observation resources are created from entities identified with the traits of `SIGN`, `SYMPTOM`, or `DIAGNOSIS`, and they are linked to the source document with this linkage resource.

For resources generated by the integrated NLP, you can make `GET` requests, but searching these new resources is not supported.

To learn more about searching these extensions using HealthLake's integration with Athena, see Query your HealthLake data store using SQL.

Contents

- Integration with the FHIR REST API operations
- Examples of how Amazon Comprehend Medical API operations are integrated into HealthLake
- Search parameters

How Amazon Comprehend Medical is integrated with HealthLake

HealthLake infers data found in the `DocumentReference` resource type using Amazon Comprehend Medical. The Amazon Comprehend Medical API operations `DetectEntities-V2`, `InferICD10-CM`, and `InferRxNorm` detect medical conditions as *traits*. Each operation provides different insights.

> ⚠ **Language support**
>
> Amazon Comprehend Medical API operations only detect medical entities in English language texts.

- **DetectEntities-V2**: Inspects the clinical text for a variety of medical entities and returns specific information about them, such as entity category, location, and confidence score.
- **InferICD10-CM**: Detects medical conditions in a patient record as entities, and it links those entities to normalized concept identifiers in the ICD-10-CM knowledge base from the CDC's National Center for Health Statistics under authorization by the World Health Organization (WHO).
- **InferRxNorm**: Detects medications as entities listed in a patient record, and it links them to the normalized concept identifiers in the RxNorm database from the National Library of Medicine.

The supported traits for each API operation are `SIGN`, `SYMPTOM`, and `DIAGNOSIS`. If traits are detected, they are added as FHIR-compliant extensions to different locations in your HealthLake data store.

Locations where extensions are added.

- `DocumentReference`: The results from the Amazon Comprehend Medical API operations are added as an `extension` to each document found within the DocumentReference resource type.

Results in the extension are divided into two groups. You can find them in the results based on their URL.

- `http://healthlake.amazonaws.com/system-generated-resources/`
  - These are resource types that have been created or added to by HealthLake.
- `http://healthlake.amazonaws.com/aws-cm/`
  - Where the raw output of the Amazon Comprehend Medical API operations is added to your HealthLake data store.

- `Linkage`: This resource type is either added or created as a result of the integrated NLP. A `GET` request on a specific `Linkage` returns a list of linked resources. To identify if a `Linkage` was added by HealthLake, look for the added `"tag": [{"display": "SYSTEM_GENERATED"}]` key-value pair. To learn more about the FHIR specifications for Linkage, see Resource type: Linkage in the *FHIR Documentation Index*.
- FHIR resource types generated as a result of the Amazon Comprehend Medical API operations.
  - `Observation`: Has results from the Amazon Comprehend Medical API operations DetectEntities-V2 and InferICD10-CM added to it when the traits are `SIGN` or `SYMPTOM`.
  - `Condition`: Has results from the Amazon Comprehend Medical API operations DetectEntities-V2 and InferICD10-CM added to it when the traits are `DIAGNOSIS`.
  - `MedicationStatement`: Has results from the Amazon Comprehend Medical API operation InferRxNorm added to it.

Integration with the FHIR REST API operations

By default, traits detected by the Amazon Comprehend Medical API operations are not returned when making a `GET` request.

To see the results of the integrated NLP operations for these resource types, you must specify a known `ID`.

- `Linkage`
- `Observation`
- `Condition`
- `MedicationStatement`

The results of the integrated NLP operations outside the DocumentReference resource type are only available using a `GET` request where the specified `ID` is know to contain results from the Amazon Comprehend Medical API operations.

Examples of how Amazon Comprehend Medical API operations are integrated into HealthLake

Example 1: Patient record ingested into a HealthLake data store

Here is an example clinical note based off of a patient's encounter with a medical professional.

> **Synthetic data**
>
> The text in this example is synthetic content and doesn't contain personal health information (PHI).

```
1991-08-31

# Chief Complaint
- Headache
- Sinus Pain
- Nasal Congestion
- Sore Throat
- Pain with Bright Lights
- Nasal Discharge
- Cough


# History of Present Illness
Jerónimo599
 is a 4 month-old non-hispanic white male.

# Social History
 Patient has never smoked.


Patient comes from a middle socioeconomic background.

Patient currently has Aetna.
```

```
# Allergies
No Known Allergies.

# Medications
No Active Medications.

# Assessment and Plan
Patient is presenting with bee venom (substance), mold (organism), house dust
 mite (organism), animal dander (substance), grass pollen (substance), tree pollen
 (substance), lisinopril, sulfamethoxazole / trimethoprim, fish (substance).

## Plan

The patient was prescribed the following medications:
- astemizole 10 mg oral tablet
- nda020800 0.3 ml epinephrine 1 mg/ml auto-injector
The patient was placed on a careplan:
- self-care interventions (procedure)
```

As a reminder, this information is encoded in base64 format in the DocumentReference resource. When this document is ingested into HealthLake and the Amazon Comprehend Medical API operations are complete, to see the results, you can start with the GETrequest on the `DocumentReference` resource type.

```
 GET https://https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/
r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/DocumentReference
```

When the Amazon Comprehend Medical API operations are successful, look for these key-value pairs inside the `extension` linked to the following `"url": "http://healthlake.amazonaws.com/aws-cm/"`

```
{
  "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."
 }
```

The following tabs show you how the ingested medical record is reported in your HealthLake data store based on the resource type.

DocumentReference

To the see the results for a single `DocumentReference` resource type, make a `GET` request where the `id` of a specific resource is provided.

```
GET https://https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/DocumentReference/0e938f03-da7f-4178-acd8-eea9586c46ed
```

When successful, you get a `200` HTTP response code, and the following JSON response (that has been truncated for clarity).

Here is the `http://healthlake.amazonaws.com/system-generated-resources/` portion. You can see that a new `Linkage/`*`e366d29f-2c22-4c19-866e-09603937935a`* has been added. You can also see where HealthLake has added inference-based findings to specific `Observation` and `Condition` resource types.

To see how these resource types have been amended, choose the related tabs.


```
{
  "extension": [
   {
    "url": "http://healthlake.amazonaws.com/linkage",
    "valueReference": {
     "reference": "Linkage/e366d29f-2c22-4c19-866e-09603937935a"
    }
   },
   {
    "url": "http://healthlake.amazonaws.com/nlp-entity",
    "valueReference": {
     "reference": "Observation/c6e0a3ff-7a17-4d8b-bfd0-d02d7da090c5"
    }
   },
   {
    "url": "http://healthlake.amazonaws.com/nlp-entity",
    "valueReference": {
     "reference": "Condition/0854e1f3-894d-448e-a8d9-3af5b9902baf"
    }
   }
```

```
  ],
  "url": "http://healthlake.amazonaws.com/system-generated-resources/"
 }
```

Linkage

To the see the results for a single `Linkage` resource type, make a `GET` request where the `ID` of a specific resource is provided.

```
GET https://https://healthlake.your-region.amazonaws.com/
datastore/your-datastore-id/r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/
Linkage/e366d29f-2c22-4c19-866e-09603937935a
```

When successful, you get a `200` HTTP response code, and the following truncated JSON response.

The response contains the `item` element. In it, the key-value pair `"type": "source"` indicates the specific `DocumentReference` entry used to modify the `Condition` and `Observations` listed under the `"type": "alternate"` key-value pair.

You also see the `meta` element, and a corresponding key-value pair, `"tag": [{"display": "SYSTEM_GENERATED"}]`, indicating these resources were created by HealthLake.

```
{
  "resourceType": "Linkage",
  "id": "e366d29f-2c22-4c19-866e-09603937935a",
  "active": true,
  "item":
  [
   {
    "type": "alternate",
    "resource": {
     "reference": "Observation/c6e0a3ff-7a17-4d8b-bfd0-d02d7da090c5",
     "type": "Observation"
    }
   },
   {
    "type": "alternate",
    "resource": {
     "reference": "Condition/9d5c1ef6-f822-4faf-b55f-7c70f2a4aa8d",
     "type": "Condition"
    }
```

```
   },
   {
    "type": "source",
    "resource": {
     "reference": "DocumentReference/0e938f03-da7f-4178-acd8-eea9586c46ed",
     "type": "DocumentReference"
    }
   }
  ],
  "meta": {
   "lastUpdated": "2022-10-21T19:38:31.327Z",
   "tag": [{
    "display": "SYSTEM_GENERATED"
   }]
  }
 }
```

Resource type: Observation

To the see the results for a single `Observation` resource type, make a `GET` request where the `ID` of a specific resource is provided.

```
GET https://https://healthlake.your-region.amazonaws.com/
datastore/your-datastore-id/r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/
Observation/e366d29f-2c22-4c19-866e-09603937935a
```

The results of the Amazon Comprehend Medical API operations are amended to the following elements: `code`, `meta`, and `modifierExtension`.

code

An element of type `CodeableConcept`. To learn more, see CodeableConcept in the *FHIR Documentation Index*.

HealthLake appends the following three key-value pairs.

- `"system": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/"`: Where the URL refers to a specific Amazon Comprehend Medical API operation. In this case, InferICD10CM.
- `"code": "A52.06"`: Where `A52.06` is the ICD-10-CM code that identifies the concept found in the knowledge base from the Centers for Disease Control.

- `"display": "Other syphilitic heart involvement"`: Where `"Other syphilitic heart involvement"` is the long description of the ICD-10-CM code in the ontology.

The following truncated JSON response contains only the `code` element.

```
"code": {
  "coding":
  [
    {
   "system": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/",
   "code": "A52.06",
   "display": "Other syphilitic heart involvement"
    }
  ],
  "text": "Other syphilitic heart involvement"
 }
```

To understand the model's confidence that the assigned ICD-10-CM code is correct, use the `modifierExtension` element.

`meta`

The `meta` element contains metadata that indicates whether the `code` element contains details that have been added by the Amazon Comprehend Medical API operations.

The following truncated JSON response contains only the `meta` element.

```
"meta": {
  "lastUpdated": "2022-10-21T19:38:30.879Z",
  "tag": [{
   "display": "SYSTEM_GENERATED"
  }]
 }
```

`modifierExtension`

The `modifierExtension` element contains more details about the level of confidence of the assigned codes found in the `code` element. It also has key-value pairs that provide a link back to the original DocumentReference used to generate the results and the related Linkage resource type.

For each `coding` element added, you will see an `entity-score` and an `entity-Concept-Score` added to the modifierExtension. For each value in the key-value pair, you see a score. For `entity-score`, this score is the level of confidence that Amazon Comprehend Medical has in the accuracy of the detection. For `entity-Concept-Score`, this score is the level of confidence that Amazon Comprehend Medical has that the entity is accurately linked to an ICD-10-CM concept.

The following truncated JSON response contains only the `modifierExtension` element.

```
"modifierExtension": [{
   "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
score",
   "valueDecimal": 0.45005733
  },
  {
   "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
Concept-Score",
   "valueDecimal": 0.1111792
  },
  {
   "url": "http://healthlake.amazonaws.com/system-generated-linkage",
   "valueReference": {
    "reference": "Linkage/e366d29f-2c22-4c19-866e-09603937935a"
   }
  },
  {
   "url": "http://healthlake.amazonaws.com/source-document-reference",
   "valueReference": {
    "reference": "DocumentReference/0e938f03-da7f-4178-acd8-eea9586c46ed"
   }
  }
 ]
```

Full JSON Response

```
{
  "subject": {
   "reference": "Patient/0679b7b7-937d-488a-b48d-6315b8e7003b"
  },
  "resourceType": "Observation",
  "status": "unknown",
  "code": {
```

```
   "coding": [{
    "system": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/",
    "code": "A52.06",
    "display": "Other syphilitic heart involvement"
   }],
   "text": "Other syphilitic heart involvement"
  },
  "meta": {
   "lastUpdated": "2022-10-21T19:38:30.879Z",
   "tag": [{
    "display": "SYSTEM_GENERATED"
   }]
  },
  "modifierExtension": [{
    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
score",
    "valueDecimal": 0.45005733
   },
   {
    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
Concept-Score",
    "valueDecimal": 0.1111792
   },
   {
    "url": "http://healthlake.amazonaws.com/system-generated-linkage",
    "valueReference": {
     "reference": "Linkage/e366d29f-2c22-4c19-866e-09603937935a"
    }
   },
   {
    "url": "http://healthlake.amazonaws.com/source-document-reference",
    "valueReference": {
     "reference": "DocumentReference/0e938f03-da7f-4178-acd8-eea9586c46ed"
    }
   }
  ],
  "id": "7e88c7c5-21a5-4dd7-8fc2-a02474fba583"
 }
```


Condition

To the see the results for a single `Condition` resource type, make a `GET` request where the `ID` of a specific resource is provided.

```
GET https://https://healthlake.your-region.amazonaws.com/datastore/your-
datastore-id/r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/Condition/b06d343d-
ddb8-4f36-82cb-853fcd434dfd
```

The results of the Amazon Comprehend Medical API operations are amended to the following elements: `code`, `meta`, and `modifierExtension`.

`code`

An element of type `CodeableConcept`. To learn more, see CodeableConcept in the *FHIR Documentation Index*.

HealthLake appends the following three key-value pairs.

- `"system": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/"`: Where the URL refers to a specific Amazon Comprehend Medical API operation. In this case, InferICD10CM.
- `"code": "I70.0"`: Where `A52.06` is the ICD-10-CM code that identifies the concept found in the knowledge base from the Centers for Disease Control.
- `"display": "Atherosclerosis of aorta"`: Where `"Other syphilitic heart involvement"` is the long description of the ICD-10-CM code in the ontology.

The following truncated JSON response contains only the `code` element.

```
"code": {
  "coding":
  [
    {
   "system": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/",
   "code": "I70.0",
   "display": "Atherosclerosis of aorta"
    }
  ],
  "text": "Atherosclerosis of aorta"
 }
```

To understand the model's confidence that the assigned ICD-10-CM code is correct, use the `modifierExtension` element.

`meta`

The `meta` element contains metadata that indicates whether the `code` element contains details that have been added by the Amazon Comprehend Medical API operations.

The following truncated JSON response contains only the `meta` element.

```
"meta": {
  "lastUpdated": "2022-10-21T19:38:30.877Z",
  "tag": [{
   "display": "SYSTEM_GENERATED"
  }]
 }
```

`modifierExtension`

The `modifierExtension` element contains more details about the level of confidence of the assigned codes found in the `code` element. It also has key-value pairs that provide a link back to the original DocumentReference used to generate the results and the related Linkage resource type.

For each `coding` element added, you will see an `entity-score` and an `entity-Concept-Score` added to the modifierExtension. For each value in the key-value pair, you see a score. For `entity-score`, this score is the level of confidence that Amazon Comprehend Medical has in the accuracy of the detection. For `entity-Concept-Score`, this score is the level of confidence that Amazon Comprehend Medical has that the entity is accurately linked to an ICD-10-CM concept.

The following truncated JSON response contains only the `modifierExtension` element.

```
"modifierExtension": [{
   "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
score",
   "valueDecimal": 0.94417894
  },
  {
   "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
Concept-Score",
   "valueDecimal": 0.8458298
  },
  {
   "url": "http://healthlake.amazonaws.com/system-generated-linkage",
   "valueReference": {
```

```
    "reference": "Linkage/e366d29f-2c22-4c19-866e-09603937935a"
   }
  },
  {
   "url": "http://healthlake.amazonaws.com/source-document-reference",
   "valueReference": {
    "reference": "DocumentReference/0e938f03-da7f-4178-acd8-eea9586c46ed"
   }
  }
 ]
```

Full JSON Response

```
{
  "subject": {
   "reference": "Patient/0679b7b7-937d-488a-b48d-6315b8e7003b"
  },
  "resourceType": "Condition",
  "code": {
   "coding": [{
    "system": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/",
    "code": "I70.0",
    "display": "Atherosclerosis of aorta"
   }],
   "text": "Atherosclerosis of aorta"
  },
  "meta": {
   "lastUpdated": "2022-10-21T19:38:30.877Z",
   "tag": [{
    "display": "SYSTEM_GENERATED"
   }]
  },
  "modifierExtension": [{
    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
score",
    "valueDecimal": 0.94417894
   },
   {
    "url": "http://healthlake.amazonaws.com/aws-cm/infer-icd10/aws-cm-icd10-entity-
Concept-Score",
    "valueDecimal": 0.8458298
   },
   {
```

```
    "url": "http://healthlake.amazonaws.com/system-generated-linkage",
    "valueReference": {
     "reference": "Linkage/e366d29f-2c22-4c19-866e-09603937935a"
    }
   },
   {
    "url": "http://healthlake.amazonaws.com/source-document-reference",
    "valueReference": {
     "reference": "DocumentReference/0e938f03-da7f-4178-acd8-eea9586c46ed"
    }
   }
  ],
  "id": "b06d343d-ddb8-4f36-82cb-853fcd434dfd"
}
```

Example 2: A `DocumentReference` that contains MedicationStatement resource type

Here is an example of a clinical note based off of a patient's encounter with a medical professional.

> **⚠ Synthetic data**
>
> The text in this example is synthetic content and doesn't contain personal health information (PHI).

```
Tom is not prescribed Advil
```

The following tabs show how the ingested medical record is reported in your HealthLake data store based on the resource type.

DocumentReference

To the see the results for a single `DocumentReference` resource type, make a `GET` request where the `ID` of a specific resource is provided.

```
GET https://https://healthlake.your-region.amazonaws.com/datastore/your-datastore-id/r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/DocumentReference/c549125d-a218-421f-b8bf-23614c5e796c
```

When successful, you get a 200 HTTP response code and the following truncated JSON response.

The key-value pair, `"url": "http://healthlake.amazonaws.com/system-generated-resources/"`, indicates that the resource types inside this `extension` have been added by Amazon Comprehend Medical API operations. You can see the new `Linkage` resource type, and multiple `MedicationStatement` resources.


```
"extension": [{
   "extension": [{
     "url": "http://healthlake.amazonaws.com/linkage",
     "valueReference": {
      "reference": "Linkage/394bb244-177b-4409-8657-26b20ed56dd7"
     }
    },
    {
     "url": "http://healthlake.amazonaws.com/nlp-entity",
     "valueReference": {
      "reference": "MedicationStatement/cbf6af10-b0b9-451c-bdde-99611e3498a8"
     }
    },
    {
     "url": "http://healthlake.amazonaws.com/nlp-entity",
     "valueReference": {
      "reference": "MedicationStatement/9a89b0d3-6681-45ca-9926-27951edce5c7"
     }
    },
    {
     "url": "http://healthlake.amazonaws.com/nlp-entity",
     "valueReference": {
      "reference": "MedicationStatement/4a01f6c8-5f3a-4122-80ab-405312f96aa2"
     }
    },
    {
     "url": "http://healthlake.amazonaws.com/nlp-entity",
     "valueReference": {
      "reference": "MedicationStatement/fbfb77d8-70cf-4579-b4c0-d6fe3c01656b"
     }
    },
    {
     "url": "http://healthlake.amazonaws.com/nlp-entity",
     "valueReference": {
      "reference": "MedicationStatement/1340c9ce-9c48-4bf9-9b2f-d0ab027f5e0b"
```

```
     }
    }
   ],
   "url": "http://healthlake.amazonaws.com/system-generated-resources/"
  }
```

Linkage

To the see the results for a single `Linkage` resource type, make a `GET` request where the `ID` of a specific resource is provided.

```
GET https://https://healthlake.your-region.amazonaws.com/
datastore/your-datastore-id/r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/
Linkage/394bb244-177b-4409-8657-26b20ed56dd7
```

When successful, you get a `200` HTTP response code and the following JSON response.

The response contains the `item` element. In it, the key-value pair `"type": "source"` indicates the specific `DocumentReference` entry used to modify the `MedicationStatement` resource types.

You can also see the `meta` element and a corresponding key-value pair, `"tag": [{"display": "SYSTEM_GENERATED"}]`, indicating that these resources were created by HealthLake.

```
{
  "resourceType": "Linkage",
  "id": "394bb244-177b-4409-8657-26b20ed56dd7",
  "active": true,
  "item": [{
    "type": "alternate",
    "resource": {
     "reference": "MedicationStatement/cbf6af10-b0b9-451c-bdde-99611e3498a8",
     "type": "MedicationStatement"
    }
   },
   {
    "type": "alternate",
    "resource": {
     "reference": "MedicationStatement/9a89b0d3-6681-45ca-9926-27951edce5c7",
     "type": "MedicationStatement"
```

```
    }
   },
   {
    "type": "alternate",
    "resource": {
     "reference": "MedicationStatement/4a01f6c8-5f3a-4122-80ab-405312f96aa2",
     "type": "MedicationStatement"
    }
   },
   {
    "type": "alternate",
    "resource": {
     "reference": "MedicationStatement/fbfb77d8-70cf-4579-b4c0-d6fe3c01656b",
     "type": "MedicationStatement"
    }
   },
   {
    "type": "alternate",
    "resource": {
     "reference": "MedicationStatement/1340c9ce-9c48-4bf9-9b2f-d0ab027f5e0b",
     "type": "MedicationStatement"
    }
   },
   {
    "type": "source",
    "resource": {
     "reference": "DocumentReference/c549125d-a218-421f-b8bf-23614c5e796c",
     "type": "DocumentReference"
    }
   }
  ],
  "meta": {
   "lastUpdated": "2022-10-24T20:05:03.501Z",
   "tag": [{
    "display": "SYSTEM_GENERATED"
   }]
  }
 }
```

MedicationStatement

To the see the results for a single `MedicationStatement` resource type, make a `GET` request where the `ID` of a specific resource is provided.

```
GET https://https://healthlake.your-region.amazonaws.com/
datastore/your-datastore-id/r4/eeb8005725ae22b35b4edbdc68cf2dfd/r4/
MedicationStatement/9a89b0d3-6681-45ca-9926-27951edce5c7
```

The MedicationStatement resource type is where the results of the Amazon Comprehend Medical InferRxNorm API operation are found. The results are amended to the following elements: `medicationCodeableConcept`, `meta`, and `modifierExtension`.

medicationCodeableConcept

An element of type `CodeableConcept`. To learn more, see CodeableConcept in the *FHIR Documentation Index*.

HealthLake appends the following three key-value pairs.

- `"system": ""http://healthlake.amazonaws.com/aws-cm/infer-rxnorm/`: Where the URL refers to a specific Amazon Comprehend Medical API operation. In this case, InferRxNorm.
- `"code": "731533"`: Where `731533` is an RxNorm concept ID, also known as the RxCUI.
- `"display": "ibuprofen 200 MG Oral Capsule [Advil]"`: Where `ibuprofen 200 MG Oral Capsule [Advil]` is the description of the RxNorm concept.

The following truncated JSON response contains only the `MedicationStatement` element.

```
"medicationCodeableConcept": {
  "coding": [
   {
    "system": "http://healthlake.amazonaws.com/aws-cm/infer-rxnorm/",
    "code": "731533",
    "display": "ibuprofen 200 MG Oral Capsule [Advil]"
   }
  ]
 }
```

meta

The `meta` element contains metadata that indicates whether the `code` element contains details that have been added by the Amazon Comprehend Medical API operations.

The following truncated JSON response contains only the `meta` element.

```
"meta": {
  "lastUpdated": "2022-10-24T20:05:02.800Z",
  "tag": [
   {
    "display": "SYSTEM_GENERATED"
   }
  ]
 }
```

`modifierExtension`

The `modifierExtension` element contains key-value pairs that provide a link back to the original DocumentReference used to generate the results and the related Linkage resource type.

```
"modifierExtension": [
   {
    "url": "http://healthlake.amazonaws.com/system-generated-linkage",
    "valueReference": {
     "reference": "Linkage/394bb244-177b-4409-8657-26b20ed56dd7"
    }
   },
   {
    "url": "http://healthlake.amazonaws.com/source-document-reference",
    "valueReference": {
     "reference": "DocumentReference/c549125d-a218-421f-b8bf-23614c5e796c"
    }
   }
 ]
```

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

Search parameters	Finds matches for
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
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 a 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 subextension that matches both the entityText and entityCategory.
detectEntities-entity-type-score	If there is at least one entity in the DetectEntities subextension that matches both the entityType and entityScore.
detectEntities-entity-text-score	If there is at least one entity in the DetectEntities subextension that matches both the entityText and entityScore.
detectEntities-entity-text-type	If there is at least one entity in the DetectEntities subextension that matches both the entityText and entityType.
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 sub-extension 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 sub-extension that matches the entityText and there is at least one conceptScore for that entity that matches the score.

Search parameters	Matches returned
infer-icd10cm-entity-concept-description-concept-score	If there is at least one concept within the entity in the InferICD10CM sub-extension that matches the concept description and the conceptScore.
infer-rxnorm-entity-text-concept-code	If there is at least one entity in the InferRxNorm sub-extension 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 sub-extension 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 sub-extension that matches the concept description and the conceptScore.

Security in AWS 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 AWS HealthLake

The AWS shared responsibility model applies to data protection in AWS 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. You are also responsible for 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 users with AWS IAM Identity Center or 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 require TLS 1.2 and recommend TLS 1.3.
- 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 sensitive 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 text 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 text 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 AWS 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 AWS HealthLake which is encrypted by that key:

```
"Statement": [
    {
        "Sid": "Allow access to create data stores and do CRUD/search in AWS
 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 AWS HealthLake

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

Identity and access management for AWS 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 AWS HealthLake identity and access.

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 service users 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 AWS HealthLake works with IAM.

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 AWS HealthLake.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might 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 Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

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:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **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 Cross account resource access in IAM 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.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
- **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 AWS 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 AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. 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**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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 an 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 AWS 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 AWS HealthLake

IAM feature	HealthLake support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	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 AWS 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 AWS HealthLake

To view examples of HealthLake identity-based policies, see Identity-based policy examples for AWS HealthLake.

Resource-based policies within AWS HealthLake

Supports resource-based policies	No

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 Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS 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 AWS 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 each action with a comma.

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

To view examples of HealthLake identity-based policies, see Identity-based policy examples for AWS HealthLake.

Policy resources for AWS 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 AWS HealthLake in the *Service Authorization Reference*. To learn the actions with which you can specify the ARN of each resource, see Actions defined by AWS HealthLake.

To view examples of HealthLake identity-based policies, see Identity-based policy examples for AWS HealthLake.

Policy condition keys for AWS HealthLake

Supports service-specific 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 AWS HealthLake in the *Service Authorization Reference*. To learn the actions and resources with which you can use a condition key, see Actions defined by AWS HealthLake.

To view examples of HealthLake identity-based policies, see Identity-based policy examples for AWS HealthLake.

Access control lists (ACLs) in AWS 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 AWS 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.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

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 AWS 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 AWS HealthLake

Supports forward access sessions (FAS)	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS 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*.

For information about service roles and the inline policy required for full access to AWS HealthLake, see Getting started with AWS HealthLake.

> **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 AWS 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 AWS 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 AWS HealthLake

By default, users and roles don't have permission to create or modify HealthLake resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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

For details about actions and resource types defined by HealthLake, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS HealthLake in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS HealthLake console

To access the AWS 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 (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 they're trying to perform.

For full access to HealthLake, attach the following policies to an IAM user or role: `AmazonHealthLakeFullAccess` and `AWSLakeFormationDataAdmin`. You also need to attach

the HealthLake inline policy which is a 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*. For information about the inline policy which creates the required service role, see Getting started with AWS HealthLake. You must also use the AWS Lake Formation console or CLI to assign your HealthLake administrator to be an AWS Lake Formation Data Lake administrator. For more information, see Getting started with AWS HealthLake.

Accessing an AWS HealthLake data store in Amazon Athena

If you want to provide users and roles with access to the HealthLake data stores in Amazon Athena, attach the following IAM policies to the role or user: `AmazonAthenaFullAccess` and `AmazonS3FullAccess`. `Select` and `Describe` permissions are also required on tables managed by AWS Lake Formation. AWS Lake Formation table permissions are granted by an AWS Lake Formation administrator in the AWS Lake Formation console or via the CLI. For more information, see Getting started with AWS HealthLake

Allowing 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": "*"
        }
    ]
}
```

AWS managed policies for AWS HealthLake

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AmazonHealthLakeFullAccess

The `AmazonHealthLakeFullAccess` policy provides full access to HealthLake. With this policy attached to their user or role, users can use HealthLake to access, query, import, and export data in HealthLake. To perform many common actions in HealthLake, you must add additional policies to the user or role. For more information, see Getting Started and HealthLake operations and permissions.

You can attach the `AmazonHealthLakeFullAccess` policy to your IAM identities.

This policy grants *`administrative and contributor`* permissions that allow users and roles to query, search, import, and export with HealthLake, and it also makes it possible for HealthLake to perform actions on behalf of the users and roles that have these permissions.

Permissions details

This policy includes the following statement.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Action": [
    "healthlake:*",
    "s3:ListAllMyBuckets",
    "s3:ListBucket",
    "s3:GetBucketLocation",
    "iam:ListRoles"
   ],
   "Resource": "*",
   "Effect": "Allow"
  },
  {
   "Effect": "Allow",
   "Action": "iam:PassRole",
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "iam:PassedToService": "healthlake.amazonaws.com"
    }
```

```
    }
   }
  ]
}
```

AWS managed policy: AmazonHealthLakeReadOnlyAccess

`AmazonHealthLakeReadOnlyAccess` policy grants read-only access and permissions to HealthLake and related resources in other AWS services. Apply this policy to users who you want to grant the ability to query and view HealthLake data stores, but not the ability to create or make changes to them.

You can attach the `AmazonHealthLakeReadOnlyAccess` policy to your IAM identities.

This policy grants *`read-only`* permissions that allow users and roles to query HealthLake.

Permissions details

This policy includes the following statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "healthlake:ListFHIRDatastores",
                "healthlake:DescribeFHIRDatastore",
                "healthlake:DescribeFHIRImportJob",
                "healthlake:DescribeFHIRExportJob",
                "healthlake:GetCapabilities",
                "healthlake:ReadResource",
                "healthlake:SearchWithGet",
                "healthlake:SearchWithPost",
                "healthlake:SearchEverything"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
```

```
}
```

HealthLake operations and permissions

The following table lists typical operations in HealthLake and the permissions needed to perform them.

HealthLake operations	Required permissions
Create a data store in HealthLake	`AmazonHealthLakeFullAccess` ,`AmazonLakeFormationDataAdmin` , inline policy, and AWS Lake Formation Administrator permissions managed by AWS Lake Formation
Delete a data store in HealthLake	`AmazonHealthLakeFullAccess` , `AmazonLakeFormationDataAdmin` , inline policy, and AWS Lake Formation Administrator permissions managed by AWS Lake Formation
List, search, or query a data store in HealthLake	`AmazonHealthLakeReadOnlyAccess`
Query a data store using Amazon Athena	`AmazonAthenaFullAccess` , `AmazonS3FullAccess` , AWS Lake Formation `Select` and `Describe` permissions on tables managed by AWS Lake Formation
Import data from HealthLake	See IAM policies for import jobs.
Export data from HealthLake	See Exporting from your data store.

HealthLake updates to AWS managed policies

View details about updates to AWS managed policies for HealthLake from the time that this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the HealthLake Document history page.

Change	Description	Date
AmazonHealthLakeFullAccess	`AmazonHealthLakeFullAccess` policy required to allow full access to HealthLake.	November, 14, 2022
AmazonHealthLakeReadOnlyAccess	`AmazonHealthLakeReadOnlyAccess` policy required for read-only access to HealthLake.	November, 14, 2022
HealthLake started tracking changes	HealthLake started tracking changes for its AWS managed policies.	November, 14, 2022

Troubleshooting AWS 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 AWS HealthLake
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS HealthLake resources

I am not authorized to perform an action in AWS 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, your policies must be updated 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 that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS 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 AWS HealthLake works with IAM.
- 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 Cross account resource access in IAM in the *IAM User Guide*.

Logging AWS HealthLake API Calls with AWS CloudTrail

AWS 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.

AWS 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 using the FHIR REST API. 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 AWS 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_GBYAZFCLLBLSUTOYYFQZRLBLQJNFOYQVRPZBOJAIIUAHICAEAGIWLNVQEYAMSXVWMBLXC
        "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 AWS HealthLake

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

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. 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 on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- 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. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- 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 AWS 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 AWS HealthLake

As a managed service, AWS 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 AWS HealthLake

AWS 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 AWS 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*.

AWS HealthLake endpoints and quotas

The following sections contain information about AWS HealthLake quotas, and endpoints. For adjustable quotas, you can request a quota increase using the Service Quotas console. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

Service endpoints

The table shows the available HealthLake service endpoints in a given region.

Region Name	Region	Endpoint	Protocol
US East (Ohio)	us-east-2	healthlake.us-east-2.amazonaws.com	HTTPS
		healthlake-fips.us-east-2.amazonaws.com	HTTPS
US East (N. Virginia)	us-east-1	healthlake.us-east-1.amazonaws.com	HTTPS
		healthlake-fips.us-east-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	healthlake.us-west-2.amazonaws.com	HTTPS
		healthlake-fips.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Mumbai)	ap-south-1	healthlake.ap-south-1.amazonaws.com	HTTPS

Service quotas for HealthLake

The following are the default quotas for HealthLake.

Name	Default	Adjustable	Description
Number of characters in a medical note	Each supported Region: 10,000	No	The maximum number of characters in an individual medical note within the DocumentReference Resource type (POST/PUT requests).
Number of concurrent StartFHIRImportJob jobs	Each supported Region: 1	No	The maximum concurrent StartFHIRImportJob jobs.
Number of concurrentStartFHIRExportJob jobs	Each supported Region: 1	No	The maximum concurrent StartFHIRExportJob jobs.
Number of data stores per account	Each supported Region: 10	Yes	The default maximum number of active data stores per account.
Number of files in a StartFHIRImportJob	Each supported Region: 10,000	No	The maximum number of files in a StartFHIRImportJob.
Number of resources per Bundle	Each supported Region: 160	No	The maximum number of resources allowed in a Bundle request.
Rate of Bundle requests per account	Each supported Region: 20	Yes	The maximum number of POST Bundle requests that you can make per second per account.
Rate of Bundle requests per data store	Each supported Region: 10	Yes	The maximum number of POST Bundle requests that you can make per

Name	Default	Adjustable	Description
			second per data store. Data stores created prior to 8/21/2023 will be limited to 1 request per second.
Rate of CancelFHIRExportJob requests using DELETE per account	Each supported Region: 1	No	The maximum number of CancelFHIRExportJob requests using DELETE that you can make per minute per account.
Rate of CreateFHIRDatastore requests per account	Each supported Region: 1	No	The maximum number of CreateFHIRDatastore requests that you can make per minute per account.
Rate of DELETE requests per account	Each supported Region: 2,000	Yes	The maximum number of DELETE requests that you can make per second per account.
Rate of DELETE requests per data store	Each supported Region: 1,000	Yes	The maximum number of DELETE requests that you can make per second per data store. Data stores created prior to 8/21/2023 will be limited to 100 requests per second.

Name	Default	Adjustable	Description
Rate of DeleteFHIRDatastore requests per account	Each supported Region: 1	No	The maximum number of DeleteFHIRDatastore requests that you can make per minute per account.
Rate of DescribeFHIRDatastore requests per account	Each supported Region: 10	No	The maximum number of DescribeFHIRDatastore requests that you can make per second per account.
Rate of DescribeFHIRExportJob requests per account	Each supported Region: 10	No	The maximum number of DescribeFHIRExportJob requests that you can make per second per account.
Rate of DescribeFHIRExportJob requests using GET per account	Each supported Region: 10	No	The maximum number of DescribeFHIRExportJob requests using GET that you can make per second per account.
Rate of DescribeFHIRImportJob requests per account	Each supported Region: 10	No	The maximum number of DescribeFHIRImportJob requests that you can make per second per account.
Rate of Discovery requests per account	Each supported Region: 10	No	The maximum number of Discovery requests that you can make per minute per account.

Name	Default	Adjustable	Description
Rate of GET requests per account	Each supported Region: 6,000	Yes	The maximum number of GET requests that you can make per second per account.
Rate of GET requests per data store	Each supported Region: 3,000	Yes	The maximum number of GET requests that you can make per second per data store. Data stores created prior to 8/21/2023 will be limited to 100 requests per second.
Rate of GetCapabilities requests per account	Each supported Region: 10	No	The maximum number of GetCapabilities requests that you can make per second per account.
Rate of ListFHIRDatastores requests per account	Each supported Region: 10	No	The maximum number of ListFHIRDatastores requests that you can make per second per account.
Rate of ListFHIRExportJobs requests per account	Each supported Region: 10	No	The maximum number of ListFHIRExportJobs requests that you can make per second per account.

Name	Default	Adjustable	Description
Rate of ListFHIRImportJobs requests per account	Each supported Region: 10	No	The maximum number of ListFHIRImportJobs requests that you can make per second per account.
Rate of ListTagsforResource requests per account	Each supported Region: 10	No	The maximum number of ListTagsforResource requests that you can make per second per account.
Rate of POST requests per account	Each supported Region: 2,000	Yes	The maximum number of POST requests that you can make per second per account.
Rate of POST requests per data store	Each supported Region: 1,000	Yes	The maximum number of POST requests that you can make per second per data store. Data stores created prior to 8/21/2023 will be limited to 100 requests per second.
Rate of PUT requests per account	Each supported Region: 2,000	Yes	The maximum number of PUT requests that you can make per second per account.

Name	Default	Adjustable	Description
Rate of PUT requests per data store	Each supported Region: 1,000	Yes	The maximum number of PUT requests that you can make per second per data store. Data stores created prior to 8/21/2023 will be limited to 100 requests per second.
Rate of StartFHIRExportJob requests per account	Each supported Region: 1	No	The maximum number of StartFHIRExportJob requests that you can make per minute per account.
Rate of StartFHIRExportJob requests using POST per account	Each supported Region: 1	No	The maximum number of StartFHIRExportJob requests using POST that you can make per minute per account.
Rate of StartFHIRImportJob requests per account	Each supported Region: 1	No	The maximum number of StartFHIRImportJob requests that you can make per minute per account.
Rate of TagResource requests per account	Each supported Region: 10	No	The maximum number of TagResource requests that you can make per second per account.

Name	Default	Adjustable	Description
Rate of UntagResource requests per account	Each supported Region: 10	No	The maximum number of UntagResource requests that you can make per second per account.
Rate of search requests using GET per account	Each supported Region: 200	Yes	The maximum number of search requests using GET that you can make per second per account.
Rate of search requests using GET per data store	Each supported Region: 100	Yes	The maximum number of search requests using GET that you can make per second per data store.
Rate of search requests using POST per account	Each supported Region: 200	Yes	The maximum number of search requests using POST that you can make per second per account.
Rate of search requests using POST per data store	Each supported Region: 100	Yes	The maximum number of search requests using POST that you can make per second per data store.
Size of individual imported file	Each supported Region: 5 Gigabytes	No	The maximum size (in GB) of an individual file included in a StartFHIR ImportJob.
Total import job size	Each supported Region: 500 Gigabytes	No	The maximum size (in GB) of all files included in the import job.

Troubleshooting

The following documentation can help you troubleshoot problems you might have with using AWS HealthLake.

Topics

Why can't I create a HealthLake data store?

On November, 14, 2022, HealthLake updated the required IAM permissions needed to create a new data store. If you haven't updated policies attached to the user or role that accesses HealthLake you get the following error.

```
AccessDeniedException: Insufficient Lake Formation permission(s): Required Database on
 Catalog
```

To view updated IAM policy requirements for creating a data store, see AWS managed policy: AmazonHealthLakeFullAccess. For step-by-step directions on how to add these policies to your IAM user or role, see Getting started with AWS HealthLake.

To create a data store, you also need use of a symmetrical customer-owned or Amazon-owned KMS key. Make sure you have the correct permissions in your IAM policy. To learn more about AWS KMS, see AWS Key Management Service in the *AWS Key Management Service Developer Guide*.

Exceeded number of data stores allowed per account

HealthLake has a quota of 10 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 AWS 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 FHIR 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 programmatically. The following sample manifest file lists the input and output Amazon S3 buckets, and also information on the number of resources scanned and how many were imported successfully.


```
            {
            "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
            }
```


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 COMPLETED_WITH_ERRORS, check the manifest file, Manifest.json, for information on which files did not import successfully.
- 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 the following:


```
{"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 not valid.

- 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.

How do I find DocumentReference resources that could not be processed?

If a DocumentReference resource was not valid, HealthLake will provide an extension indicating a validation error instead of the integrated medical NLP output. In order to find DocumentReference resources that led to a validation error during NLP processing, customers can use HealthLake's search function with search key **cm-decoration-status** and search value **VALIDATION_ERROR**. This search will list all DocumentReference resources that led to validation errors, along with an error message describing the nature of the error. The structure of the extension field in those DocumentReference resources with validation errors will resemble the following example.

```
"extension": [
          {
              "extension": [
                  {
                      "url": "http://healthlake.amazonaws.com/aws-cm/status/",
                      "valueString": "VALIDATION_ERROR"
                  },
                  {
                      "url": "http://healthlake.amazonaws.com/aws-cm/message/",
                      "valueString": "Resource led to too many nested objects after NLP
 operation processed the document. 10937 nested objects exceeds the limit of 10000."
                  }
              ],
              "url": "http://healthlake.amazonaws.com/aws-cm/"
          }
    ]
```

A **VALIDATION_ERROR** can also occur if NLP decoration creates more than 10,000 nested objects. When this happens, the document must be split into smaller documents before processing.

Migrating an existing data store to use Amazon Athena

data stores created before November, 14, 2022 are functional, but are not queryable in Athena using SQL. To query a preexisting data store with Athena, you must first migrate it to a new data store.

To migrate data to a new data store

1. Create a new data store.
2. Export the data from the pre-existing to an Amazon S3 bucket.
3. Import the data into the new data store from the Amazon S3 bucket.

Exporting data to an Amazon S3 bucket incurs an extra charge. The extra charge depends on the size of the data that you export.

Connecting search results in Athena to other AWS services

You can experience issues when sharing your search results from Athena with other AWS services.

Issue can occur when you use `json_extract[1]` as part of a SQL search query.

To fix this issue, you must update to CATVAR.

You might encounter this issue when trying to **Create** save results, a **Table** (static), or **View** (dynamic).

The Athena console is not working after importing data into a new data store

After you import data into a new data store, the data may not be available for use immediately. This is to allow time for the data to be ingested into the iceberg tables. Try again at a later time.

Why do I get a Lake Formation permissions error: lakeformation:PutDataLakeSettings when adding a new data lake administrator?

If your IAM user or role contains the `AWSLakeFormationDataAdmin` AWS managed policy you cannot add new data lake administrators. You will get an error containing the following:

```
User arn:aws:sts::111122223333:assumed-role/lakeformation-admin-user is not authorized
 to perform: lakeformation:PutDataLakeSettings on resource: arn:aws:lakeformation:us-
east-2:111122223333:catalog:111122223333 with an explicit deny in an identity-based
 policy
```

The AWS managed policy `AdministratorAccess` is required to add an IAM user or role as a AWS Lake Formation data lake administrator. If your IAM user or role also contains `AWSLakeFormationDataAdmin` the action will fail. The `AWSLakeFormationDataAdmin` AWS managed policy contains an explict deny for the AWS Lake Formation API operation, `PutDataLakeSetting`.

Even administrators with full access to AWS using the `AdministratorAccess` AWS managed policy can be limited by the `AWSLakeFormationDataAdmin` policy.

How do I turn on HealthLake's integrated natural language processing feature?

As of February 20, 2023, the default behavior of HealthLake data stores changed.

Current data stores: All current HealthLake data stores will stop using natural language processing (NLP) on base64-encoded `DocumentReference` resources. This means that new `DocumentReference` resources will not be analyzed using NLP, and no new resources will be generated based off of text in the `DocumentReference` resource type. For existing `DocumentReference` resources, the data and resources generated via NLP remain, but they will not be updated after February 20, 2023.

New data stores: HealthLake data stores created after February 20, 2023 will *not* perform natural language processing (NLP) on base64-encoded `DocumentReference` resources.

To turn on this feature you must create a case using AWS Support Center Console. To create your case, log in to your AWS account, and then choose **Create case**. To learn more about creating a case

and case management, see Creating support cases and case management in the *AWS Support User Guide*.

My data store status is not changing from Creating

If you try to create a new HealthLake data store, and your data store status is not changing from **Creating** you need to update Athena to use the AWS Glue Data Catalog.

To learn more, see Upgrading to the AWS Glue Data Catalog step-by-step in the *Amazon Athena User Guide*.

After successfully upgrading the AWS Glue Data Catalog, you can now create a data store.

To remove the old data store get started by creating a case using AWS Support Center Console. To create your case, log in to your AWS account, and then choose **Create case**. To learn more, see Creating support cases and case management in the *AWS Support User Guide*.

My SDK data store creation status returns an exception or unknown status

Please update your SDK to the latest version if your list data store or describe data store API calls return an exception or unknown data store status.

My FHIR POST API operation with a 10MB document to HealthLake gets a 413Request Entity Too Large error.

AWS HealthLake has a synchronous Create and Update API limit of 5MB to avoid increased latencies and timeouts.

You can ingest large documents, upto 164MB, using the Binary ResourceType using the Bulk Import API.

Document History for the AWS HealthLake Developer Guide

The following table describes the documentation changes for AWS HealthLake releases.

- **API version:** latest
- **Latest documentation update:** 12/09/2023

Change	Description	Date
HealthLake now supports new FHIR search parameters, extension and resource type.	HealthLake now supports new FHIR search parameters, extension and resource type.	December 9, 2023
HealthLake now supports the SMART on FHIR framework	HealthLake now supports creating SMART on FHIR enabled HealthLake data stores.	May 31, 2023
HealthLake now supports profile validation	HealthLake now supports FHIR profile validation.	May 31, 2023
HealthLake now supports `export`	HealthLake now supports exporting files using the FHIR REST API operation `export`.	May 31, 2023
Asia Pacific (Mumbai) region	AWS HealthLake is now available in the Asia Pacific (Mumbai) region.	April 4, 2023
Integrated natural language processing turned off	HealthLake turned off integrated natural language processing (NLP) on all data stores as of February 20, 2023.	February 20, 2023

HealthLake integrates with Amazon Athena	You can now use Athena to query data stores created after November, 14, 2022.	November 14, 2022
Total import job size increased	Maximum total size of all files in a StartFHIRImportJob request is now 500 GB.	October 3, 2022
Bundle support	HealthLake now supports the Bundle resource type for ingesting multiple resources.	August 5, 2022
Updated Quotas for CRUD operations in HealthLake	HealthLake now supports higher limits for CRUD requests.	July 14, 2022
Include support	HealthLake now supports `_include` in data store queries.	July 14, 2022
AWS HealthLake is now generally available	HealthLake is now generally available.	July 30, 2020

AWS Glossary

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