



User Guide

# AWS HealthOmics



**Version latest**

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

# AWS HealthOmics: User Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

---

# Table of Contents

<b>What is AWS HealthOmics?</b> .....	<b>1</b>
Important notice .....	1
Concepts .....	2
Storage .....	2
Analytics .....	2
Workflows .....	3
HealthOmics features .....	3
Related services .....	4
Regions and endpoints for AWS HealthOmics .....	5
How to access HealthOmics .....	5
Learn more .....	5
<b>Setting up HealthOmics</b> .....	<b>7</b>
Sign up for an AWS account .....	7
Create a user with administrative access .....	7
<b>Getting started</b> .....	<b>9</b>
Getting Started (API) .....	9
Creating a sequence store .....	9
Creating a variant store .....	10
Creating a workflow .....	11
Getting Started (Console) .....	15
HealthOmics Storage .....	15
HealthOmics Workflows .....	17
<b>HealthOmics Storage</b> .....	<b>18</b>
ETag calculation and data provenance .....	19
How ETags are calculated .....	20
Creating reference and sequence stores .....	21
Creating and managing reference stores .....	21
Creating and managing sequence stores .....	26
Deleting reference and sequence stores .....	28
Sequence store imports .....	28
Direct upload to a sequence store .....	39
Exporting read sets .....	45
Accessing and managing read sets with Amazon S3 URIs .....	47
Amazon S3 URI structure in HealthOmics storage .....	48

Using Hosted or Local IGV to access read sets .....	49
Using Samtools or HTSlib in HealthOmics .....	49
Using Mountpoint HealthOmics .....	50
Using CloudFront with HealthOmics .....	50
Activating read sets .....	50
<b>HealthOmics Analytics .....</b>	<b>54</b>
Creating variant stores .....	54
Creating variant stores using the console .....	55
Creating variant stores using the API .....	55
Creating variant store import jobs .....	57
Creating annotation stores .....	61
Create an annotation store using the console .....	61
Create an annotation store using the API .....	62
Creating new versions of annotation stores .....	63
Creating annotation store import jobs .....	67
Create an annotation import job using the API .....	67
Additional parameters for TSV and VCF formats .....	69
Creating TSV formatted annotation stores .....	70
Starting VCF formatted import jobs .....	73
Deleting analytics stores .....	74
Querying analytics data .....	74
Setting up Lake Formation .....	75
Configuring Athena for queries .....	78
Running queries .....	78
Sharing analytics stores .....	80
Creating a store share .....	80
<b>HealthOmics Workflows .....</b>	<b>82</b>
Ready2Run workflows .....	82
Using Ready2Run workflows (console) .....	83
Using Ready2Run workflows (API) .....	84
Private workflows .....	85
Setting up Amazon ECR .....	86
Writing workflow definition files .....	89
Creating private workflows .....	104
Sharing workflows .....	114
Creating run groups .....	118

Running workflows .....	120
Run storage types .....	121
Starting a workflow run .....	123
Deleting workflows and runs .....	129
Define custom IAM permissions for runs .....	129
Using the CloudWatch Logs for troubleshooting .....	131
<b>Resource sharing .....</b>	<b>132</b>
Create a share .....	132
Retrieve information about a share .....	133
View the shares that you own .....	134
View accepted shares from other accounts .....	134
Delete a share .....	134
<b>Tagging resources in HealthOmics .....</b>	<b>135</b>
Important notice .....	135
Tagging HealthOmics resources .....	135
Best practices .....	136
Tagging requirements .....	137
Adding a tag to an HealthOmics resource .....	137
Listing tags for a resource .....	138
Removing tags from a data store .....	139
<b>Permissions .....</b>	<b>140</b>
User policies .....	140
Service roles .....	142
Sample IAM policies .....	142
Sample CloudWatch templates .....	144
Resource permissions .....	146
Lake Formation permissions .....	146
Amazon ECR permissions .....	147
Amazon S3 URI Permissions .....	147
<b>Security .....</b>	<b>149</b>
Data protection .....	149
Encryption at rest .....	150
Identity and access management .....	159
Audience .....	159
Authenticating with identities .....	160
Managing access using policies .....	163

How AWS HealthOmics works with IAM .....	166
Identity-based policy examples .....	175
AWS managed policies .....	177
Troubleshooting .....	181
Compliance validation .....	183
Resilience .....	184
VPC endpoints (AWS PrivateLink) .....	184
Considerations for HealthOmics VPC endpoints .....	184
Creating an interface VPC endpoint for HealthOmics .....	185
Creating a VPC endpoint policy for HealthOmics .....	185
Special considerations for accessing read sets using Amazon S3 URIs .....	186
<b>Monitoring AWS HealthOmics .....</b>	<b>188</b>
CloudWatch .....	189
Viewing AWS HealthOmics metrics .....	189
Creating an alarm .....	190
CloudTrail logs .....	190
HealthOmics information in CloudTrail .....	191
Understanding HealthOmics log file entries .....	192
EventBridge .....	193
EventBridge event message structure and examples .....	194
<b>Troubleshooting .....</b>	<b>198</b>
Why can't I run my workflow? .....	198
Why do I get a "not a currently supported operation" error when running Nextflow? .....	199
Why can't I create a reference store? .....	199
Why can't I create a sequence store? .....	199
Why can't I create a workflow? .....	199
Why did my task fail? .....	199
Why can't I import my BAM, CRAM or FASTQ files? .....	200
Why can't I import my VCF or gVCF files? .....	200
Why can't I see my annotation store or variant store in Athena? .....	200
Why can't I access my data store in Athena? .....	200
Why do I get a "Request Too Long" error message when I try to create a workflow? .....	200
Error and status messages for run failures .....	201
<b>Quotas .....</b>	<b>204</b>
Service quotas .....	204
File size quotas .....	209

---

API quotas .....	213
<b>Document history .....</b>	<b>216</b>

# What is AWS HealthOmics?

AWS HealthOmics is an AWS service that helps users such as bioinformaticians, researchers, and scientists to store, query, analyze, and generate insights from genomics and other biological data. It simplifies and accelerates the process of storing and analyzing genomic information for research and clinical organizations, and makes scientific discovery and insight generation faster.

HealthOmics has three primary components. HealthOmics Storage helps you store and share petabytes of genomics data efficiently and at low cost per gigabase. HealthOmics Analytics simplifies how you prepare genomics data for multiomics and multimodal analyses. HealthOmics Workflows automatically provisions and scales the underlying infrastructure for your bioinformatics computation.

## Topics

- [Important notice](#)
- [HealthOmics concepts](#)
- [HealthOmics features](#)
- [Related services](#)
- [Regions and endpoints for AWS HealthOmics](#)
- [How to access HealthOmics](#)
- [Learn more](#)

## Important notice

HealthOmics isn't a substitute for professional medical advice, diagnosis, or treatment, and isn't intended to cure, treat, mitigate, prevent, or diagnose any disease or health condition. You are responsible for instituting human review as part of any use of AWS HealthOmics, including in association with any third-party product intended to inform clinical decision-making.

HealthOmics is intended only for the transferring, storing, formatting, or displaying of data, and for the provision of infrastructure and configuration support for managing workflows. AWS HealthOmics isn't intended to directly perform variant calling or genomic analysis and interpretation. AWS HealthOmics isn't intended to interpret or analyze clinical laboratory tests or other device data, results, and findings, and isn't a substitute for third-party tools intended for use in genomic analyses.



# HealthOmics concepts

This topic covers definitions for key concepts and terms that are specific to HealthOmics, to help you understand the terminology of HealthOmics used in this guide.

## Topics

- [Storage](#)
- [Analytics](#)
- [Workflows](#)

## Storage

Data storage is separated into sequence stores, for your genomics sequences and related information, and a reference store, for all of your reference genomes. The following terms describe the implementations that are specific to HealthOmics.

- *Sequence store* – A data store for the storage of genomics files. You can have one or more sequence stores within HealthOmics. Access permissions and AWS KMS encryption can be set on a sequence store to control who has access to the data.
- *Read set* – A read set is an abstraction of genomics reads, which are stored in FASTQ, BAM, or CRAM formats. Read sets can be imported into sequence stores and annotated with metadata. You can apply permissions to read sets using attribute based access control (ABAC).
- *Reference* – A genome reference is used with reads to identify where in a genome a specific read, or group of reads, is mapped to. These are in FASTA format and stored in the reference store.
- *Reference store* – A data store for the storage of reference genomes. You can have a single reference store in each account and region.

## Analytics

You can transform and analyze your genomics data with HealthOmics Analytics. Create a variant store or annotation store to include additional information for your queries.

- *Variant store* – data store that stores variant data at a population scale. Variant stores support both genomic Variant Call Format (gVCF) and VCF inputs.

- *Annotation store* – A data store representing an annotation database, such as one from a TSV/CSV, VCF, or General Feature Format (GFF3) file. Annotation Stores are mapped to the same coordinate system as variant stores during an import.

## Workflows

With HealthOmics Workflows, you can process and analyze your genomics data.

- *Workflow* – The overall definition of an end to end process including parameters and references to tools. Workflow definitions can be expressed as WDL, Nextflow, or CWL. Each created workflow has a unique identifier.
- *Run/Workflow run* – A single invocation of a workflow. An individual run uses your defined input data and produces an output. Each created run has a unique identifier.
- *Task* – The individual processes within a run. HealthOmics Workflows use these defined compute specifications to run your task. Each task has a unique identifier.
- *Run group* – A group of runs for which you can set the max vCPU, max duration, or max concurrent runs to help limit the compute resources used per run. You can specify and configure priorities for your workflow runs within a run group. For example, you can specify that a high priority run will be performed before one that's lower priority, creating a priority queue. It is optional to use a Run Group, and each Run Group has a unique identifier.

## HealthOmics features

HealthOmics offers the following features.

- HealthOmics Storage — helps you store and share petabytes of raw genomics data efficiently and at low cost per gigabase.
- HealthOmics Analytics — simplifies how you prepare genomics data for multiomics and multimodal analyses.
- HealthOmics Workflows — automatically provisions and scales the underlying infrastructure for your bioinformatics workflows.

You can use each component independently, or as part of an integrated end-to-end solution.

HealthOmics offers you the following benefits.

- Securely store and combine genomic data — HealthOmics integrates with other AWS services such as AWS Lake Formation and Amazon Athena. You can securely store your genomics data and then query or combine it with medical history data for better diagnoses and personalized treatment plans.
- Protect patient privacy — HealthOmics is HIPAA eligible. It also integrates with IAM and Amazon CloudWatch so that you can control and log data access, and track how the data is used in analyses.
- Built to scale — Support large population data analyses with simplified billing and new collaboration tools.
- Maximize efficiency — Use automated workflows and integrated tools to streamline data processing and analysis.

You can use HealthOmics for the following biomedical applications:

- Population sequencing — Query thousands of genomes at once to understand how genomic variation maps to phenotypes across a population.
- Clinical genomics — Build reproducible genomics workflows from sequencer output to reportable data. You can also optimize for high volume throughput and set the compute requirements for high-priority clinical samples to reduce turnaround time.
- Clinical trials — Integrate genome analysis into clinical trials to better understand the efficacy of new drug candidates. Simplify and accelerate clinical trials with long-term cost savings and data provenance to meet regulations from governing bodies.
- Enhance research and innovation — Streamline and control storage, access, and analysis of anonymized genomics data with built-in row and column-based access control.

## Related services

The following services work with HealthOmics.

- Amazon Elastic Container Registry – Each private workflow uses an Amazon ECR image (in a private Amazon ECR repository) to contain all executables, libraries, and scripts required to run the workflow.
- Amazon Simple Storage Service – Amazon S3 provides file storage for Store and Workflow data.
- AWS Lake Formation – Lake Formation manages data access to your Analytics data stores.
- Amazon Athena – Use Athena to perform queries on your Variant stores.

- Amazon SageMaker – Use SageMaker to run HealthOmics tasks using Jupyter notebooks.

## Regions and endpoints for AWS HealthOmics

For a full list of regions and endpoints, see the [AWS General Reference](#).

In addition to the AWS regions that are active by default, there are also *Opt-in Regions* which need to be activated. To learn more about how to activate or deactivate a Region, see [Specify which AWS Regions your account can use](#) in the AWS Account Management guide.

## How to access HealthOmics

You can access AWS HealthOmics features using the management console, CLI, SDKs or API.

- AWS Management Console – Provides a web interface that you can use to access HealthOmics.
- AWS Command Line Interface (AWS CLI) – Provides commands for a broad set of AWS services, including AWS HealthOmics, and is supported on Windows, macOS, and Linux. For more information about installing the AWS CLI, see [AWS Command Line Interface](#).
- AWS SDKs – AWS provides SDKs (Software Development Kits) that consist of libraries and sample code for various programming languages and platforms (including Java, Python, Ruby, .NET, iOS, and Android). The SDKs provide a convenient way to use HealthOmics programmatically. For more information, see the [AWS SDK Developer Center](#).
- AWS API – You can use API operations to access and manage HealthOmics programmatically. For more information, see the [HealthOmics API Reference](#).

## Learn more

Learn more about HealthOmics from these workshops and tutorials:

- HealthOmics workshop – [HealthOmics end to end workshop](#)
- AWS genomics resources – [Public Amazon ECR repositories](#) related to genomics
- Python tutorials – [Jupyter notebook tutorials](#) on Github, covering HealthOmics storage, analytics, and workflows

Become familiar with additional HealthOmics tools that AWS provides:

- WDL linter – [HealthOmics linter for WDL](#)
- Nextflow linter – [HealthOmics linter for Nextflow](#)
- HealthOmics Amazon ECR helper tool – [Amazon ECR helper tool for HealthOmics](#)
- HealthOmics tools on Github – [Tools for working with HealthOmics](#) (Transfer manager, URI parser, Omics rerun, Run analyzer).

# Setting up HealthOmics

## Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

### To sign up for 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](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

## Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

### Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

## Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

## Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

## Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

# Getting started with HealthOmics

The following topics help you learn the concepts of each component of HealthOmics. You can start using the service by creating a data store or setting up a basic workflow.

## Topics

- [Getting Started \(API\)](#)
- [Getting Started \(Console\)](#)

## Getting Started (API)

The following examples are intended to help you start using the service by creating a data store or setting up a basic workflow. As prerequisites, your data must be in an Amazon S3 bucket in same region (for example, us-west-2) and the most recent version of the AWS CLI installed.

## Topics

- [Creating a sequence store](#)
- [Creating a variant store](#)
- [Creating a workflow](#)

## Creating a sequence store

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

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

HealthOmics Storage provides storage for genomic files in FASTQ, BAM, and CRAM formats. These files are stored in Read Sets, which are an AWS resource. To store Read Sets, you need to first create a sequence store, as shown in the following example.

```
aws omics create-sequence-store --name "MySequenceStore"
```



You will receive the following response in JSON, which include the ID number for your newly created sequence store.

```
{
  "id": "3936421177",
  "arn": "arn:aws:omics:us-west-2:(account):sequenceStore/3936421177",
  "name": "MySequenceStore",
  "creationTime": "2022-07-13T20:09:26.038Z"
}
```

## Creating a variant store

The following example demonstrates using the `CreateVariantStore` operation with the AWS CLI. To run the example, you must install the latest version of the AWS CLI.

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

To create a variant store, we will need a `referenceName` and `name` parameter. The variant store is ready to ingest data when its status is shown as `READY`.

```
aws omics create-variant-store --name (storeName) --reference (referenceArn)
```

To confirm the creation of your variant store, you will receive the following response.

```
{
  "id": "b533f097bade",
  "reference": {
    "referenceArn": "arn:aws:omics:us-
west-2:451654099157:referenceStore/5638433913/reference/5871590330"
  },
  "status": "CREATING",
  "name": "variantstore",
  "creationTime": "2022-11-08T01:29:36.594566+00:00"
}
```

## Creating a workflow

You will need both input and output Amazon S3 buckets, as well as an IAM role with access to those buckets. The following is an example IAM policy that grants permission to access the contents of an Amazon S3 bucket, the Amazon ECR containers, and Cloud Watch logs.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::[s3path]/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::[s3path]"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::[output_s3path]/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ]
    }
  ]
}
```

```

    ],
    "Resource": [
      "arn:aws:logs:{{region}}:{{accountId}}:log-group:/aws/omics/
WorkflowLog:log-stream:*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup"
    ],
    "Resource": [
      "arn:aws:logs:{{region}}:{{accountId}}:log-group:/aws/omics/
WorkflowLog:*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "ecr:BatchGetImage",
      "ecr:GetDownloadUrlForLayer",
      "ecr:BatchCheckLayerAvailability"
    ],
    "Resource": [
      "arn:aws:ecr:{{region}}:{{accountId}}:repository/*"
    ]
  }
]
}

```

The role will need to authorize the service to assume it before it can be used in a workflow run. This can be done by adding "trust relationships" similar to the following statement.

```

{
  "Effect": "Allow",
  "Principal": {
    "Service": "omics.amazonaws.com"
  },
  "Action": "sts:AssumeRole"
}

```

Workflow definitions must be written in the supported languages, either WDL or Nextflow. The following is a basic example that demonstrates a workflow that reads the contents of an INPUT file and writes them into a RESULT file.

```
version 1.0

# Simple demo workflow, copy input file into output file

workflow TestFlow {
  input {
    File input_txt_file
  }

  #copies input file data to output.
  call TxtFileCopyTask{
    input:
      input_txt_file = input_txt_file,
  }

  output {
    File output_txt_file = TxtFileCopyTask.output_txt_file
  }
}

#Task Definitions

task TxtFileCopyTask {
  input {
    File input_txt_file
  }

  command {
    cat ~{input_txt_file} > outfile.txt
  }

  output {
    File output_txt_file = "outfile.txt"
  }
}
```

The workflow definition files need to be zipped before calling the HealthOmics CreateWorkflow API operation.

```
zip definition.zip main.wdl
```

Define your parameters with a parameter-template file like the following JSON file.

```
{
  "image": {
    "description": "Optional ECR image",
    "optional": true
  },
  "file": {
    "description": "Required input file"
  }
}
```

Once you've defined your workflow and the parameters, you can create a workflow using the API as shown.

```
aws omics create-workflow --name Sample --description BasicExample --definition-zip
fileb://definition.zip --parameter-template file://params_sample_description.json
```

Once you've created your workflow, you should receive the following response to confirm that the workflow has been created.

```
{
  "arn": "arn:aws:omics:us-west-2:....",
  "id": "12345",
  "status": "CREATING",
  "tags": {
    "resourceArn": "arn:aws:omics:us-west-2:...."
  }
}
```

# Getting Started (Console)

The **Getting started** page of the console covers the three main HealthOmics components: Genomics data storage, Workflows, and Analytics. The following examples will help you start using each component.

## Topics

- [HealthOmics Storage](#)
- [HealthOmics Workflows](#)

## HealthOmics Storage

With HealthOmics Storage, you can create a sequence or reference store, import a reference genome, and import genomics files. After you create your stores and import your genomic data, you can access and analyze your sequence data.

## Topics

- [Create a reference store](#)
- [Create a sequence store](#)
- [Import genomics files](#)

## Create a reference store

### To create a reference store

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Get started with HealthOmics**.
3. Choose **Reference genomes** from the Genomics data storage options.
4. You can either choose a previously imported reference genome or import a new one. If you haven't imported a reference genome, choose **Import reference genome** in the top right.
5. On the **Create reference genome import job** page, choose either the **Quick create** or **Manual create** option to create a reference store, and then provide the following information.
  - **Reference genome name** - A unique name for this store.
  - **Description** (optional) - A description of this reference store.

- **IAM Role** - Select a role with access to your reference genome.
- **Reference from Amazon S3** - Select your reference sequence file in an Amazon S3 bucket.
- **Tags** (optional) - Provide up to 50 tags for this reference store.

## Create a sequence store

### To create a sequence store

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Sequence stores**.
3. On the **Create sequence store** page, provide the following information
  - **Sequence store name** - A unique name for this store.
  - **Description** (optional) - A description of this sequence store.
  - **Data Encryption** - Select whether you want data encryption to be owned and managed by AWS or to use a customer managed CMK.
  - **Tags** (optional) - Provide up to 50 tags for this sequence store.

## Import genomics files

### To import a genomics file

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Sequence stores**.
3. On the **Sequence stores** page, choose the sequence store that you want to import your files into.
4. On the individual sequence store page, choose **Import genomic files**.
5. On the **Specify import details** page, provide the following information
  - **IAM role** - The IAM role that can access the genomic files on Amazon S3.
  - **Reference genome** - The reference genome for this genomics data.
6. On the **Specify import manifest** page, specify the following information **Manifest file**. The manifest file is a JSON or YAML file that describes essential information of your genomics data. For information about the manifest file, see [Sequence store imports](#).

7. Click **Create import job**.

## HealthOmics Workflows

The following exercise shows how to use a Ready2Run workflow. A Ready2Run workflow is preconfigured with the parameters and tool references you need to run the workflow. The workflow publisher provides sample data, so that you can try out the workflow without creating your own data.

### To use a Ready2Run workflow in the console

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Ready2Run workflows**.
3. On the **Ready2Run workflows** page, choose the **ESMFold for up to 800 residues** workflow. The console opens the details page for that workflow.
4. The details tab provides information about the workflow. To try out the workflow, choose **Create run**.
5. In the **Specify run details** page, enter a run name.
6. Enter or select an Amazon S3 location for the workflow run output.
7. For **Run meta data retention mode**, choose whether to retain or remove run meta data.
8. In the **Service role** panel, choose **Create and use a new service role**.
9. Choose **Next**.
10. From the **Add parameters** page, choose **Run workflow with Ready2Run test data**.
11. Choose **Next**.
12. Review your inputs, then choose **Start run**.



# HealthOmics Storage

Use HealthOmics Storage to store, retrieve, organize, and share genomics data efficiently and at low cost. HealthOmics Storage understands the relationships between different data objects, so that you can define which read sets originated from the same source data. This provides you with data provenance.

Data that's stored in ACTIVE state is retrievable immediately. Data that hasn't been accessed for 30 days or more is stored in ARCHIVE state. To access archived data, you can reactivate it through the API operations or console.

With the HealthOmics Storage API operations, you can perform the following actions:

- Create, manage, and delete sequence and reference stores
- Create and manage read sets
- Import, export, and work with read sets
- Share and access read sets with collaborators through Amazon S3 URI access
- Create, manage, and import references
- Copy read sets to local file systems for analysis
- Tag AWS resources such as sequence stores, read sets, and references
- List and read files through Amazon S3 API operations by using the Amazon S3 URI

HealthOmics sequence stores are designed to preserve the content integrity of files. However, bitwise equivalence of imported data files and exported files isn't preserved because of the compression during active and archive tiering.

During ingestion, HealthOmics generates an entity tag, or *HealthOmics ETag*, to make it possible to validate the content integrity of your data files. Sequencing portions are identified and captured as an ETag at the source level of a read set. The ETag calculation doesn't alter the actual file or genomic data. After a read set is created, the ETag shouldn't change throughout the lifecycle of the read set source. This means that reimporting the same file results in the same ETag value being calculated.

## ETag calculation and data provenance

An HealthOmics entity tag or HealthOmics ETag is a hash of the ingested content in a sequence store. This simplifies data retrieval and processing while maintaining the content integrity of the ingested data files. The ETag reflects changes to the semantic content of the object, not its metadata. The specified read set type and algorithm determine how the ETag is calculated. The ETag calculation doesn't alter the actual file or genomic data. When the file type schema of the read set permits it, the sequence store updates fields that are linked to data provenance.

Files have a bitwise identity and a semantic identity. The bitwise identity means that the bits of a file are identical, and a semantic identity means that the contents of a file are identical. Semantic identity is resilient to metadata changes and compression changes as it captures the content integrity of the file.

Read sets in HealthOmics sequence stores undergo compression/decompression cycles and data provenance tracking throughout an object's lifecycle. During this processing, the bitwise identity of an ingested file may change and is expected to change each time a file is activated; however, the semantic identity of the file is maintained. The semantic identity is captured as a HealthOmics entity tag, or ETag that's calculated during sequence store ingestion and available as read set metadata.

An HealthOmics entity tag or HealthOmics ETag is a hash of the ingested content's semantic identity in a sequence store. This simplifies data retrieval and processing, while maintaining the content integrity of the ingested data files. The ETag reflects changes to the semantic content of the object, not its metadata. The specified read set type and algorithm determine how the ETag is calculated. The ETag calculation doesn't alter the actual file or genomic data.

When the file type schema of the read set permits it, the sequence store updates fields that are linked to data provenance. For uBAM, BAM, and CRAM files, a new @CO or Comment tag is added to the header. The comment contains the sequence store ID and ingestion timestamp.

When accessing a file using the Amazon S3 URI, Amazon S3 API operations may also return Amazon S3 ETag and checksum values. The Amazon S3 ETag and checksum values differ from the HealthOmics ETags because they represent the file's bitwise identity. To learn more about descriptive metadata and Objects, see the Amazon S3 [API Object documentation](#). Amazon S3 ETag values can change with each activation cycle of a read set and you can use them to validate the reading of a file. However, don't cache Amazon S3 ETag values to use for file identity validation during the file's lifecycle because they don't remain consistent. In contrast, the HealthOmics ETag remains consistent throughout the read set's lifecycle.

## How ETags are calculated

The ETag is generated from a hash of the ingested file contents. The ETag algorithm family is set to MD5up by default, but it can be configured differently during sequence store creation. When the ETag is calculated, the algorithm and the calculated hashes are added to the read set. The supported MD5 algorithms for file types are as follows.

- *FASTQ\_MD5up* – Calculates the MD5 hash of an uncompressed, complete FASTQ read set source.
- *BAM\_MD5up* – Calculates the MD5 hash of the alignment section of an uncompressed BAM or uBAM read set source as represented in the SAM, based on the linked reference, if one is available.
- *CRAM\_MD5up* – Calculates the MD5 hash of the alignment section of the uncompressed CRAM read set source as represented in the SAM, based on the linked reference.

### Note

MD5 hashing is known to be vulnerable to collisions. Because of this, two different files might have the same ETag if they were manufactured to exploit the known collision.

The following algorithms are supported for the SHA256 family. The algorithms are calculated as follows:

- *FASTQ\_SHA256up* – Calculates the SHA-256 hash of an uncompressed, complete FASTQ read set source.
- *BAM\_SHA256up* – Calculates the SHA-256 hash of the alignment section of an uncompressed BAM or uBAM read set source as represented in the SAM, based on the linked reference, if one is available.
- *CRAM\_SHA256up* – Calculates the SHA-256 hash of the alignment section of an uncompressed CRAM read set source as represented in the SAM, based on the linked reference.

The following algorithms are supported for the SHA512 family. The algorithms are calculated as follows:

- *FASTQ\_SHA512up* – Calculates the SHA-512 hash of an uncompressed, complete FASTQ read set source.

- *BAM\_SHA512up* – Calculates the SHA-512 hash of the alignment section of an uncompressed BAM or uBAM read set source as represented in the SAM, based on the linked reference, if one is available.
- *CRAM\_SHA512up* – Calculates the SHA-512 hash of the alignment section of an uncompressed CRAM read set source as represented in the SAM, based on the linked reference.

## Creating reference and sequence stores

Reference and sequence stores are AWS resources that you can use to manage your genomic data through the API, AWS CLI, and console. The first step is for you to create a reference store to hold reference genomes that are used to map your read sets.

### Creating and managing reference stores

The following example shows you how to create a reference store by using the AWS CLI. You can have one reference store per AWS Region. Reference stores support storage of FASTA files with the extensions `.fasta`, `.fa`, `.fas`, `.fsa`, `.faa`, `.fna`, `.ffn`, `.frn`, `.mpfa`, `.seq`, `.txt`. The bgzip version of these extensions is also supported. In the following example, replace *reference store name* with the name you've chosen for your reference store.

```
aws omics create-reference-store --name "reference store name"
```

You receive a JSON response with the reference store ID and name, the ARN, and the timestamp of when your reference store was created.

```
{
  "id": "3242349265",
  "arn": "arn:aws:omics:us-west-2:555555555555:referenceStore/3242349265",
  "name": "MyReferenceStore",
  "creationTime": "2022-07-01T20:58:42.878Z"
}
```

You can use the reference store ID in additional AWS CLI commands. You can retrieve the list of reference store IDs linked to your account by using the **list-reference-stores** command, as shown in the following example.

```
aws omics list-reference-stores
```

In response, you receive the name of your newly created reference store.

```
{
  "referenceStores": [
    {
      "id": "3242349265",
      "arn": "arn:aws:omics:us-west-2:555555555555:referenceStore/3242349265",
      "name": "MyReferenceStore",
      "creationTime": "2022-07-01T20:58:42.878Z"
    }
  ]
}
```

After you create a reference store, you can create import jobs to load genomic reference files into it. To do so, you must use or create an IAM role to access the data. The following is an example policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
      ]
    }
  ]
}
```

You must also have a trust policy similar to the following example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
        "Effect": "Allow",
        "Principal": {
            "Service": [
                "omics.amazonaws.com"
            ]
        },
        "Action": "sts:AssumeRole"
    }
]
```

You can now import a reference genome. This example uses Genome Reference Consortium Human Build 38 (hg38), which is open access and available from the [Registry of Open Data on AWS](#). The bucket that hosts this data is based in US East (Ohio). To use buckets in other AWS Regions, you can copy the data to an Amazon S3 bucket hosted in your Region. Use the following AWS CLI command to copy the genome to your Amazon S3 bucket.

```
aws s3 cp s3://broad-references/hg38/v0/Homo_sapiens_assembly38.fasta s3://DOC-EXAMPLE-BUCKET
```

You can then begin your import job. Replace *reference store ID*, *role ARN*, and *source file path* with your own input.

```
aws omics start-reference-import-job --reference-store-id reference store ID --role-arn role ARN --sources source file path
```

After the data is imported, you receive the following response in JSON.

```
{
    "id": "7252016478",
    "referenceStoreId": "3242349265",
    "roleArn": "arn:aws:iam::111122223333:role/OmicsReferenceImport",
    "status": "CREATED",
    "creationTime": "2022-07-01T21:15:13.727Z"
}
```

You can monitor the status of a job by using the following command. In the following example, replace *reference store ID* and *job ID* with your reference store ID and the job ID that you want to learn more about.

```
aws omics get-reference-import-job --reference-store-id reference store ID --id job ID
```

In response, you receive a response with the details for that reference store and its status.

```
{
  "id": "7252016478",
  "referenceStoreId": "3242349265",
  "roleArn": "arn:aws:iam::555555555555:role/OmicsReferenceImport",
  "status": "RUNNING",
  "creationTime": "2022-07-01T21:15:13.727Z",
  "sources": [
    {
      "sourceFile": "s3://DOC-EXAMPLE-BUCKET/Homo_sapiens_assembly38.fasta",
      "status": "IN_PROGRESS",
      "name": "MyReference"
    }
  ]
}
```

You can also find the reference that was imported by listing your references and filtering them based on the reference name. Replace *reference store ID* with your reference store ID, and add an optional filter to narrow the list.

```
aws omics list-references --reference-store-id reference store ID --filter
name=MyReference
```

In response, you receive the following information.

```
{
  "references": [
    {
      "id": "1234567890",
      "arn": "arn:aws:omics:us-west-2:555555555555:referenceStore/1234567890/
reference/1234567890",
      "referenceStoreId": "12345678",
      "md5": "7ff134953dcca8c8997453bbb80b6b5e",
      "status": "ACTIVE",
      "name": "MyReference",
      "creationTime": "2022-07-02T00:15:19.787Z",
      "updateTime": "2022-07-02T00:15:19.787Z"
    }
  ]
}
```

```
    }  
  ]  
}
```

To learn more about the reference metadata, use the **get-reference-metadata** API operation. In the following example, replace *reference store ID* with your reference store ID and *reference ID* with the reference ID that you want to learn more about.

```
aws omics get-reference-metadata --reference-store-id reference store ID --id reference ID
```

You receive the following information in response.

```
{  
  "id": "1234567890",  
  "arn": "arn:aws:omics:us-west-2:555555555555:referenceStore/referencestoreID/  
reference/referenceID",  
  "referenceStoreId": "1234567890",  
  "md5": "7ff134953dcca8c8997453bbb80b6b5e",  
  "status": "ACTIVE",  
  "name": "MyReference",  
  "creationTime": "2022-07-02T00:15:19.787Z",  
  "updateTime": "2022-07-02T00:15:19.787Z",  
  "files": {  
    "source": {  
      "totalParts": 31,  
      "partSize": 104857600,  
      "contentLength": 3249912778  
    },  
    "index": {  
      "totalParts": 1,  
      "partSize": 104857600,  
      "contentLength": 160928  
    }  
  }  
}
```

You can also download parts of the reference file by using **get-reference**. In the following example, replace *reference store ID* with your reference store ID and *reference ID* with the reference ID that you want to download from.



```
aws omics get-reference --reference-store-id reference store ID --id reference ID --  
part-number 1 outfile.fa
```

## Creating and managing sequence stores

HealthOmics sequence stores support storage of genomic files in the unaligned formats of FASTQ (gzip-only) and uBAM. It also supports the aligned formats of BAM and CRAM. Imported files are stored as read sets, which are an AWS resource. This means that you can add tags and control access through IAM. Aligned read sets require a reference genome to align genomic sequences, but it's optional for unaligned read sets.

To store read sets, you first create a sequence store. When you create a sequence store, you can specify an optional Amazon S3 bucket as a fallback location. The fallback location is used for storing any files that fail to create a read set during a direct upload. Fallback locations are available for sequence stores created after May 15, 2023. You specify the fallback location when you create the sequence store. You can't add a fallback location after the sequence store is created.

In the following example, replace *sequence store name* with the name you chose for your sequence store.

```
aws omics create-sequence-store --name sequence store name --fallback-location "s3://  
DOC-EXAMPLE-BUCKET"
```

You receive the following response in JSON, which includes the ID number for your newly created sequence store.

```
{  
  "id": "3936421177",  
  "arn": "arn:aws:omics:us-west-2:111122223333:sequenceStore/3936421177",  
  "name": "sequence_store_example_name",  
  "creationTime": "2022-07-13T20:09:26.038Z"  
  "fallbackLocation" : "s3://DOC-EXAMPLE-BUCKET"  
}
```

You can also view all sequence stores associated with your account by using the **list-sequence-stores** command, as shown in the following.

```
aws omics list-sequence-stores
```

You receive the following response.

```
{
  "sequenceStores": [
    {
      "arn": "arn:aws:omics:us-west-2:111122223333:sequenceStore/3936421177",
      "id": "3936421177",
      "name": "MySequenceStore",
      "creationTime": "2022-07-13T20:09:26.038Z"
      "fallbackLocation" : "s3://DOC-EXAMPLE-BUCKET"
    }
  ]
}
```

Additionally, you can use **get-sequence-store** to learn more about a sequence store by using its ID, as shown in the following.

```
aws omics get-sequence-store --id sequence store ID
```

```
{
  "arn": "arn:aws:omics:us-west-2:123456789012:sequenceStore/sequencestoreID",
  "creationTime": "2024-01-12T04:45:29.857Z",
  "description": null,
  "fallbackLocation": null,
  "id": "2015356892",
  "name": "MySequenceStore",
  "s3Access": {
    "s3AccessPointArn": "arn:aws:s3:us-west-2:123456789012:accesspoint/592761533288-2015356892",
    "s3Uri": "s3://592761533288-2015356892-ajdpi90jdas90a79fh9a8ja98jdfa9j9f98-s3alias/592761533288/sequenceStore/2015356892/"
  },
  "sseConfig": {
    "keyArn": "arn:aws:kms:us-west-2:123456789012:key/eb2b30f5-635d-4b6d-b0f9-d3889fe0e648",
    "type": "KMS"
  }
}
```

## Deleting reference and sequence stores

Both reference and sequence stores can be deleted. Sequence stores can only be deleted if they don't contain read sets, and reference stores can only be deleted if they don't contain references. Deleting a sequence or reference store also deletes any tags associated with that store.

The following example shows how to delete a reference store by using the AWS CLI. If the action is successful, you won't receive a response. In the following example, replace *reference store ID* with your reference store ID.

```
aws omics delete-reference-store --id reference store ID
```

The following example shows you how to delete a sequence store. You don't receive a response if the action succeeds. In the following example, replace *sequence store ID* with your sequence store ID.

```
aws omics delete-sequence-store --id sequence store ID
```

You can also delete a reference in a reference store as shown in the following example. References can only be deleted if they aren't being used in a read set, variant store, or annotation store. In the following example, replace *reference store ID* with your reference store ID, and replace *reference ID* with the ID for the reference you want to delete.

```
aws omics delete-reference --id reference ID --reference-store-id reference store ID
```

## Sequence store imports

After you create your sequence store, you can create import jobs to load files into the data store. You can upload your files from an Amazon S3 bucket, or you can upload directly by using the synchronous API operations. Your Amazon S3 bucket must be in the same Region as your sequence store. You can use a command similar to the following to move files into your Amazon S3 bucket.

You can upload any combination of aligned and unaligned read sets into your sequence store, however, if any of the read sets in your import are aligned, you must include a reference genome.

```
aws s3 cp s3://1000genomes/phase1/data/HG00100/alignment/  
HG00100.chrom20.ILLUMINA.bwa.GBR.low_coverage.20101123.bam s3://your-bucket
```

```
aws s3 cp s3://1000genomes/phase3/data/HG00146/sequence_read/SRR233106_1.filt.fastq.gz
s3://your-bucket
aws s3 cp s3://1000genomes/phase3/data/HG00146/sequence_read/SRR233106_2.filt.fastq.gz
s3://your-bucket
aws s3 cp s3://1000genomes/data/HG00096/alignment/
HG00096.alt_bwamem_GRCh38DH.20150718.GBR.low_coverage.cram s3://your-bucket
aws s3 cp s3://gatk-test-data/wgs_ubam/NA12878_20k/NA12878_A.bam s3://your-bucket
```

The sample BAM and CRAM used in this example require different genome references, Hg19 and Hg38. To learn more or to access these references, see [The Broad Genome References](#) in the Registry of Open Data on AWS.

You can reuse the IAM access policy that you used to create the Reference store. You must also create a manifest file in JSON to model the import job in `import.json` (see the following example). If you are creating a sequence store in the console, you don't have to specify the `sequenceStoreId` or `roleArn`, so your manifest file will start with the `sources` input.

## API manifest

This example code is used to import three read sets by using the API: one FASTQ, one BAM, and one CRAM.

```
{
  "sequenceStoreId": "3936421177",
  "roleArn": "arn:aws:iam::555555555555:role/OmicsImport",
  "sources":
  [
    {
      "sourceFiles":
      {
        "source1": "s3://DOC-EXAMPLE-BUCKET/
HG00100.chrom20.ILLUMINA.bwa.GBR.low_coverage.20101123.bam"
      },
      "sourceFileType": "BAM",
      "subjectId": "mySubject",
      "sampleId": "mySample",
      "referenceArn": "arn:aws:omics:us-
west-2:555555555555:referenceStore/0123456789/reference/0000000001",
      "name": "HG00100",
      "description": "BAM for HG00100",
      "generatedFrom": "1000 Genomes"
    }
  ]
}
```

```
    },
    {
      "sourceFiles":
      {
        "source1": "s3://DOC-EXAMPLE-BUCKET/SRR233106_1.filt.fastq.gz",
        "source2": "s3://DOC-EXAMPLE-BUCKET/SRR233106_2.filt.fastq.gz"
      },
      "sourceFileType": "FASTQ",
      "subjectId": "mySubject",
      "sampleId": "mySample",
      // NOTE: there is no reference arn required here
      "name": "HG00146",
      "description": "FASTQ for HG00146",
      "generatedFrom": "1000 Genomes"
    },
    {
      "sourceFiles":
      {
        "source1": "s3://DOC-EXAMPLE-BUCKET/
HG00096.alt_bwamem_GRCh38DH.20150718.GBR.low_coverage.cram"
      },
      "sourceFileType": "CRAM",
      "subjectId": "mySubject",
      "sampleId": "mySample",
      "referenceArn": "arn:aws:omics:us-
west-2:555555555555:referenceStore/0123456789/reference/0000000001",
      "name": "HG00096",
      "description": "CRAM for HG00096",
      "generatedFrom": "1000 Genomes"
    },
    {
      "sourceFiles":
      {
        "source1": "s3://DOC-EXAMPLE-BUCKET/NA12878_A.bam"
      },
      "sourceFileType": "UBAM",
      "subjectId": "mySubject",
      "sampleId": "mySample",
      // NOTE: there is no reference arn required here
      "name": "NA12878_A",
      "description": "uBAM for NA12878",
      "generatedFrom": "GATK Test Data"
    }
  ]
}
```

```
}
```

## Console manifest

This example code is used to import a single read set by using the console.

```
[
  {
    "sourceFiles":
    {
      "source1": "s3://DOC-EXAMPLE-BUCKET/
HG00100.chrom20.ILLUMINA.bwa.GBR.low_coverage.20101123.bam"
    },
    "sourceFileType": "BAM",
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "name": "HG00100",
    "description": "BAM for HG00100",
    "generatedFrom": "1000 Genomes"
  },
  {
    "sourceFiles":
    {
      "source1": "s3://DOC-EXAMPLE-BUCKET/SRR233106_1.filt.fastq.gz",
      "source2": "s3://DOC-EXAMPLE-BUCKET/SRR233106_2.filt.fastq.gz"
    },
    "sourceFileType": "FASTQ",
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "name": "HG00146",
    "description": "FASTQ for HG00146",
    "generatedFrom": "1000 Genomes"
  },
  {
    "sourceFiles":
    {
      "source1": "s3://your-bucket/
HG00096.alt_bwamem_GRCh38DH.20150718.GBR.low_coverage.cram"
    },
    "sourceFileType": "CRAM",
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "name": "HG00096",
    "description": "CRAM for HG00096",
```

```
    "generatedFrom": "1000 Genomes"
  },
  {
    "sourceFiles":
    {
      "source1": "s3://DOC-EXAMPLE-BUCKET/NA12878_A.bam"
    },
    "sourceFileType": "UBAM",
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "name": "NA12878_A",
    "description": "uBAM for NA12878",
    "generatedFrom": "GATK Test Data"
  }
]
```

Alternatively, you can upload the manifest file in YAML format.

To start the import job, use the following AWS CLI command.

```
aws omics start-read-set-import-job --cli-input-json file://import.json
```

You receive the following response, which indicates successful job creation.

```
{
  "id": "3660451514",
  "sequenceStoreId": "3936421177",
  "roleArn": "arn:aws:iam::111122223333:role/OmicsImport",
  "status": "CREATED",
  "creationTime": "2022-07-13T22:14:59.309Z"
}
```

After the import job starts, you can monitor its progress with the following command. In the following example, replace *sequence store id* with your sequence store ID, and replace *job import ID* with the import ID.

```
aws omics get-read-set-import-job --sequence-store-id sequence store id --id job import ID
```

The following shows the statuses for all import jobs associated with the specified sequence store ID.

```

{
  "id": "1234567890",
  "sequenceStoreId": "1234567890",
  "roleArn": "arn:aws:iam::111122223333:role/OmicsImport",
  "status": "RUNNING",
  "statusMessage": "The job is currently in progress."
  "creationTime": "2022-07-13T22:14:59.309Z",
  "sources": [
    {
      "sourceFiles":
      {
        "source1": "s3://DOC-EXAMPLE-BUCKET/
HG00100.chrom20.ILLUMINA.bwa.GBR.low_coverage.20101123.bam"
      },
      "sourceFileType": "BAM",
      "status": "IN_PROGRESS",
      "statusMessage": "The job is currently in progress."
      "subjectId": "mySubject",
      "sampleId": "mySample",
      "referenceArn": "arn:aws:omics:us-
west-2:111122223333:referenceStore/3242349265/reference/8625408453",
      "name": "HG00100",
      "description": "BAM for HG00100",
      "generatedFrom": "1000 Genomes"
    },
    {
      "sourceFiles":
      {
        "source1": "s3://DOC-EXAMPLE-BUCKET/SRR233106_1.filt.fastq.gz",
        "source2": "s3://DOC-EXAMPLE-BUCKET/SRR233106_2.filt.fastq.gz"
      },
      "sourceFileType": "FASTQ",
      "status": "IN_PROGRESS",
      "statusMessage": "The job is currently in progress."
      "subjectId": "mySubject",
      "sampleId": "mySample",
      "name": "HG00146",
      "description": "FASTQ for HG00146",
      "generatedFrom": "1000 Genomes"
    },
    {
      "sourceFiles":
      {

```



```

        "source1": "s3://DOC-EXAMPLE-BUCKET/
HG00096.alt_bwamem_GRCh38DH.20150718.GBR.low_coverage.cram"
    },
    "sourceFileType": "CRAM",
    "status": "IN_PROGRESS",
    "statusMessage": "The job is currently in progress."
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "referenceArn": "arn:aws:omics:us-
west-2:111122223333:referenceStore/3242349265/reference/1234568870",
    "name": "HG00096",
    "description": "CRAM for HG00096",
    "generatedFrom": "1000 Genomes"
},
{
    "sourceFiles":
    {
        "source1": "s3://DOC-EXAMPLE-BUCKET/NA12878_A.bam"
    },
    "sourceFileType": "UBAM",
    "status": "IN_PROGRESS",
    "statusMessage": "The job is currently in progress."
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "name": "NA12878_A",
    "description": "uBAM for NA12878",
    "generatedFrom": "GATK Test Data"
}
]
}

```

After the job completes, you can use the **list-read-sets** API operation to find the imported sequence files. In the following example, replace *sequence store id* with your sequence store ID.

```
aws omics list-read-sets --sequence-store-id sequence store id
```

You receive the following response.

```
{
  "readSets": [
    {
```

```
    "id": "0000000001",
    "arn": "arn:aws:omics:us-west-2:111122223333:sequenceStore/01234567890/
readSet/0000000001",
    "sequenceStoreId": "1234567890",
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "status": "ACTIVE",
    "name": "HG00100",
    "description": "BAM for HG00100",
    "referenceArn": "arn:aws:omics:us-
west-2:111122223333:referenceStore/01234567890/reference/0000000001",
    "fileType": "BAM",
    "sequenceInformation": {
      "totalReadCount": 9194,
      "totalBaseCount": 928594,
      "generatedFrom": "1000 Genomes",
      "alignment": "ALIGNED"
    },
    "creationTime": "2022-07-13T23:25:20Z"
  },
  "creationType": "IMPORT",
  "etag": {
    "algorithm": "BAM_MD5up",
    "source1": "d1d65429212d61d115bb19f510d4bd02"
  }
},
{
  "id": "0000000002",
  "arn": "arn:aws:omics:us-west-2:111122223333:sequenceStore/0123456789/
readSet/0000000002",
  "sequenceStoreId": "0123456789",
  "subjectId": "mySubject",
  "sampleId": "mySample",
  "status": "ACTIVE",
  "name": "HG00146",
  "description": "FASTQ for HG00146",
  "fileType": "FASTQ",
  "sequenceInformation": {
    "totalReadCount": 8000000,
    "totalBaseCount": 1184000000,
    "generatedFrom": "1000 Genomes",
    "alignment": "UNALIGNED"
  },
  "creationTime": "2022-07-13T23:26:43Z"
  "creationType": "IMPORT",
```

```

    "etag": {
      "algorithm": "FASTQ_MD5up",
      "source1": "ca78f685c26e7cc2bf3e28e3ec4d49cd"
    }
  },
  {
    "id": "0000000003",
    "arn": "arn:aws:omics:us-west-2:111122223333:sequenceStore/0123456789/
readSet/0000000003",
    "sequenceStoreId": "0123456789",
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "status": "ACTIVE",
    "name": "HG00096",
    "description": "CRAM for HG00096",
    "referenceArn": "arn:aws:omics:us-
west-2:111122223333:referenceStore/0123456789/reference/0000000001",
    "fileType": "CRAM",
    "sequenceInformation": {
      "totalReadCount": 85466534,
      "totalBaseCount": 24000004881,
      "generatedFrom": "1000 Genomes",
      "alignment": "ALIGNED"
    },
    "creationTime": "2022-07-13T23:30:41Z"
    "creationType": "IMPORT",
    "etag": {
      "algorithm": "CRAM_MD5up",
      "source1": "66817940f3025a760e6da4652f3e927e"
    }
  },
  {
    "id": "0000000004",
    "arn": "arn:aws:omics:us-west-2:111122223333:sequenceStore/0123456789/
readSet/0000000004",
    "sequenceStoreId": "0123456789",
    "subjectId": "mySubject",
    "sampleId": "mySample",
    "status": "ACTIVE",
    "name": "NA12878_A",
    "description": "uBAM for NA12878",
    "fileType": "UBAM",
    "sequenceInformation": {
      "totalReadCount": 20000,

```

```

        "totalBaseCount": 5000000,
        "generatedFrom": "GATK Test Data",
        "alignment": "ALIGNED"
    },
    "creationTime": "2022-07-13T23:30:41Z"
  "creationType": "IMPORT",
  "etag": {
    "algorithm": "BAM_MD5up",
    "source1": "640eb686263e9f63bcda12c35b84f5c7"
  }
}
]
}

```

To view more details about a given read set, use the **get-read-set-metadata** API operation. In the following example, replace *sequence store id* with your sequence store ID, and replace *read set id* with your read set ID.

```
aws omics get-read-set-metadata --sequence-store-id sequence store id --id read set id
```

You receive the following response.

```

{
  "arn": "arn:aws:omics:us-west-2:123456789012:sequenceStore/2015356892/readSet/9515444019",
  "creationTime": "2024-01-12T04:50:33.548Z",
  "creationType": "IMPORT",
  "description": null,
  "etag": {
    "algorithm": "FASTQ_MD5up",
    "source1": "00d0885ba3eeb211c8c84520d3fa26ec",
    "source2": "00d0885ba3eeb211c8c84520d3fa26ec"
  },
  "fileType": "FASTQ",
  "files": {
    "index": null,
    "source1": {
      "contentLength": 10818,
      "partSize": 104857600,
      "s3Access": {

```

```

    "s3Uri": "s3://accountID-sequence store ID-ajdpi90jdas90a79fh9a8ja98jdfa9jf98-
s3alias/592761533288/sequenceStore/2015356892/readSet/9515444019/
import_source1.fastq.gz"
  },
  "totalParts": 1
},
"source2": {
  "contentLength": 10818,
  "partSize": 104857600,
  "s3Access": {
    "s3Uri": "s3://accountID-sequence store ID-ajdpi90jdas90a79fh9a8ja98jdfa9jf98-
s3alias/592761533288/sequenceStore/2015356892/readSet/9515444019/
import_source1.fastq.gz"
  },
  "totalParts": 1
}
},
"id": "9515444019",
"name": "paired-fastq-import",
"sampleId": "sampleId-paired-fastq-import",
"sequenceInformation": {
  "alignment": "UNALIGNED",
  "generatedFrom": null,
  "totalBaseCount": 30000,
  "totalReadCount": 200
},
"sequenceStoreId": "2015356892",
"status": "ACTIVE",
"statusMessage": null,
"subjectId": "subjectId-paired-fastq-import"
}

```

You can use **get-read-set** to download in parallel by downloading individual parts. These parts are similar to Amazon S3 parts. The following is an example of how to download part 1 from a read set. In the following example, replace *sequence store id* with your sequence store ID, and replace *read set id* with your read set ID.

```
aws omics get-read-set --sequence-store-id sequence store id --id read set id --part-
number 1 outfile.bam
```

You can also use the HealthOmics Transfer Manager to download files for a HealthOmics reference or read set. You can download the HealthOmics Transfer Manager [here](#). For more information about using and setting up the Transfer Manager, see this [GitHub Repository](#).

## Direct upload to a sequence store

The HealthOmics Transfer Manager is recommended for adding files to your sequence store. You can also upload your read sets directly to a sequence store through the direct upload API operations.

Direct upload read sets exist first in `PROCESSING_UPLOAD` state. This means that the file parts are currently being uploaded, and you can access the read set metadata. After the parts are uploaded and the checksums are validated, the read set becomes `ACTIVE` and behaves the same as an imported read set.

If the direct upload fails, the read set status is shown as `UPLOAD_FAILED`. You can configure an Amazon S3 bucket as a fallback location for any files that fail to upload. The file parts for those read sets are transferred to the fallback location. Fallback locations are available on sequence stores created after May 15, 2023. You must also have an IAM policy that grants you read access to that Amazon S3 location.

To begin, start a multipart upload. You can do this by using the AWS CLI, as shown in the following example.

First, you create the parts by separating your data, as shown in the following example.

```
split -b 100MiB SRR233106_1.filt.fastq.gz source1_part_
```

After your source files are in parts, you can then create a multipart read set upload, as shown in the following. In the following example, replace *sequence store ID* and the other parameters with your sequence store ID and other values.

```
aws omics create-multipart-read-set-upload \  
  --sequence-store-id sequence store ID \  
  --name upload name \  
  --source-file-type FASTQ \  
  --subject-id subject ID \  
  --sample-id sample ID \  
  --description "FASTQ for HG00146" "description of upload" \  
  --
```

```
--generated-from "1000 Genomes""source of imported files"
```

In the response, you will get the uploadID and other metadata. Use the uploadID for the next step of the upload process.

```
{
  "sequenceStoreId": "1504776472",
  "uploadId": "7640892890",
  "sourceFileType": "FASTQ",
  "subjectId": "mySubject",
  "sampleId": "mySample",
  "generatedFrom": "1000 Genomes",
  "name": "HG00146",
  "description": "FASTQ for HG00146",
  "creationTime": "2023-11-20T23:40:47.437522+00:00"
}
```

Next, add your read sets to the upload. If your file is small enough, you only have to perform this step once. For larger files, you perform this step for each part of your file. If you upload a new part by using a previously used part number, it overwrites the previously uploaded part.

In the following example, replace *sequence store ID*, *upload ID*, and the other parameters with your values.

```
aws omics upload-read-set-part \
  --sequence-store-id sequence store ID \
  --upload-id upload ID \
  --part-source SOURCE1 \
  --part-number part number \
  --payload source1/source1_part_aa.fastq.gz
```

The response is an ID that you can use to verify that the uploaded file matches the file you intended.

```
{
  "checksum": "984979b9928ae8d8622286c4a9cd8e99d964a22d59ed0f5722e1733eb280e635"
}
```

Continue uploading the parts of your file, if necessary. To verify that your read sets have been uploaded, use the **list-read-set-upload-parts** API operation, as shown in the following. In the

following example, replace *sequence store ID* , *upload ID*, and the *part source* with your own input.

```
aws omics list-read-set-upload-parts \  
  --sequence-store-id sequence store ID \  
  --upload-id upload ID \  
  --part-source SOURCE1
```

The response returns the number of read sets, the size, and the timestamp for when it was most recently updated.

```
{  
  "parts": [  
    {  
      "partNumber": 1,  
      "partSize": 104857600,  
      "partSource": "SOURCE1",  
      "checksum": "MVMQk+vB9C3Ge8ADHkbKq752n3BCUzy141qEkq10D5M=",  
      "creationTime": "2023-11-20T23:58:03.500823+00:00",  
      "lastUpdatedTime": "2023-11-20T23:58:03.500831+00:00"  
    },  
    {  
      "partNumber": 2,  
      "partSize": 104857600,  
      "partSource": "SOURCE1",  
      "checksum": "keZzVzJNChAqg0dZMv0mjBwr0PM0enPj1UAfs0nvRto=",  
      "creationTime": "2023-11-21T00:02:03.813013+00:00",  
      "lastUpdatedTime": "2023-11-21T00:02:03.813025+00:00"  
    },  
    {  
      "partNumber": 3,  
      "partSize": 100339539,  
      "partSource": "SOURCE1",  
      "checksum": "TBkNfMsaeDpXzEf3ldlbi0ipFDPaohKHyZ+LF1J4CHK=",  
      "creationTime": "2023-11-21T00:09:11.705198+00:00",  
      "lastUpdatedTime": "2023-11-21T00:09:11.705208+00:00"  
    }  
  ]  
}
```

To view all active multipart read set uploads, use **list-multipart-read-set-uploads**, as shown in the following. Replace *sequence store ID* with the ID for your own sequence store.



```
aws omics list-multipart-read-set-uploads --sequence-store-id sequence store ID
```

This API only returns multipart read set uploads that are in progress. After the ingested read sets are ACTIVE, or if the upload has failed, the upload will not be returned in the response to the **list-multipart-read-set-uploads** API. To view active read sets, use the **list-read-sets** API. An example response for **list-multipart-read-set-uploads** is shown in the following.

```
{
  "uploads": [
    {
      "sequenceStoreId": "1234567890",
      "uploadId": "8749584421",
      "sourceFileType": "FASTQ",
      "subjectId": "mySubject",
      "sampleId": "mySample",
      "generatedFrom": "1000 Genomes",
      "name": "HG00146",
      "description": "FASTQ for HG00146",
      "creationTime": "2023-11-29T19:22:51.349298+00:00"
    },
    {
      "sequenceStoreId": "1234567890",
      "uploadId": "5290538638",
      "sourceFileType": "BAM",
      "subjectId": "mySubject",
      "sampleId": "mySample",
      "generatedFrom": "1000 Genomes",
      "referenceArn": "arn:aws:omics:us-
west-2:123456789012:referenceStore/8168613728/reference/2190697383",
      "name": "HG00146",
      "description": "BAM for HG00146",
      "creationTime": "2023-11-29T19:23:33.116516+00:00"
    },
    {
      "sequenceStoreId": "1234567890",
      "uploadId": "4174220862",
      "sourceFileType": "BAM",
      "subjectId": "mySubject",
      "sampleId": "mySample",
      "generatedFrom": "1000 Genomes",
```

```
        "referenceArn": "arn:aws:omics:us-  
west-2:123456789012:referenceStore/8168613728/reference/2190697383",  
        "name": "HG00147",  
        "description": "BAM for HG00147",  
        "creationTime": "2023-11-29T19:23:47.007866+00:00"  
    }  
]  
}
```

After you upload all parts of your file, use **complete-multipart-read-set-upload** to conclude the upload process, as shown in the following. Replace *sequence store ID*, *upload ID*, and the parameter for parts with your own values.

```
aws omics complete-multipart-read-set-upload \  
  --sequence-store-id sequence store ID \  
  --upload-id upload ID \  
  --parts '["checksum":"gaCBQMe+rpCFZxLpoP6gydBoXaKKDA/  
Vobh5zBDb4W4=", "partNumber":1, "partSource":"SOURCE1"]']'
```

The response for **complete-multipart-read-set-upload** is the read set IDs for your imported read sets.

```
{  
  "readSetId": "0000000001"  
}
```

To stop the upload, use **abort-multipart-read-set-upload** with the upload ID to end the upload process. Replace *sequence store ID* and *upload ID* with your own parameter values.

```
aws omics abort-multipart-read-set-upload \  
  --sequence-store-id sequence store ID \  
  --upload-id upload ID
```

After the upload is complete, you can retrieve your data from the read set by using **get-read-set**, as shown in the following. If the upload is still processing, **get-read-set** returns limited metadata, and the generated index files are unavailable. Replace *sequence store ID* and the other parameters with your own input.

```
aws omics get-read-set  
  --sequence-store-id sequence store ID \  
  --read-set-id read set ID
```

```
--id read set ID \  
--file SOURCE1 \  
--part-number 1 myfile.fastq.gz
```

To check the metadata, including the status of your upload, use the **get-read-set-metadata** API operation.

```
aws omics get-read-set-metadata --sequence-store-id sequence store ID --id read set ID
```

The response includes metadata details such as the file type, the reference ARN, the number of files, and the length of the sequences. It also includes the status. Possible statuses are `PROCESSING_UPLOAD`, `ACTIVE`, and `UPLOAD_FAILED`.

```
{  
  "id": "0000000001",  
  "arn": "arn:aws:omics:us-west-2:555555555555:sequenceStore/0123456789/  
readSet/0000000001",  
  "sequenceStoreId": "0123456789",  
  "subjectId": "mySubject",  
  "sampleId": "mySample",  
  "status": "PROCESSING_UPLOAD",  
  "name": "HG00146",  
  "description": "FASTQ for HG00146",  
  "fileType": "FASTQ",  
  "creationTime": "2022-07-13T23:25:20Z",  
  "files": {  
    "source1": {  
      "totalParts": 5,  
      "partSize": 123456789012,  
      "contentLength": 6836725,  
    },  
    "source2": {  
      "totalParts": 5,  
      "partSize": 123456789056,  
      "contentLength": 6836726,  
    }  
  },  
  "creationType": "UPLOAD"  
}
```

If a read set creation fails, the files can be moved to a fallback Amazon S3 location. This way, you can keep the files in Amazon S3 to re-import after the files issues are resolved. The fallback location can be configured for a sequence store from the console, the AWS CLI, or the API.

## Exporting read sets

You can export read sets as a batch export job to an Amazon S3 bucket. To do so, first create an IAM policy that has write access to the bucket, similar to the following IAM policy example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
      ]
    }
  ]
}
```

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

After the IAM policy is in place, begin your read set export job. The following example shows you how to do this by using the **start-read-set-export-job** API operation. In the following example, replace all parameters, such as *sequence store ID*, *destination*, *role ARN*, and *sources*, with your input.

```
aws omics start-read-set-export-job
  --sequence-store-id sequence store id \
  --destination valid s3 uri \
  --role-arn role ARN \
  --sources readSetId=read set id_1 readSetId=read set id_2
```

You receive the following response with information on the origin sequence store and the destination Amazon S3 bucket.

```
{
  "id": <job-id>,
  "sequenceStoreId": <sequence-store-id>,
  "destination": <destination-s3-uri>,
  "status": "SUBMITTED",
  "creationTime": "2022-10-22T01:33:38.079000+00:00"
}
```

After the job starts, you can determine its status by using the **get-read-set-export-job** API operation, as shown in the following. Replace the *sequence store ID* and *job ID* with your sequence store ID and job ID, respectively.

```
aws omics get-read-set-export-job --id job-id --sequence-store-id sequence store ID
```

You can view all export jobs initialized for a sequence store by using the **list-read-set-export-jobs** API operation, as shown in the following. Replace the *sequence store ID* with your sequence store ID.

```
aws omics list-read-set-export-jobs --sequence-store-id sequence store ID.
```

```
{
  "exportJobs": [
    {
      "id": <job-id>,
      "sequenceStoreId": <sequence-store-id>,

```

```
    "destination": <destination-s3-uri>,\n    "status": "COMPLETED",\n    "creationTime": "2022-10-22T01:33:38.079000+00:00",\n    "completionTime": "2022-10-22T01:34:28.941000+00:00"\n  }\n]\n}
```

In addition to exporting your read sets, you can also share them by using the Amazon S3 access URIs. To learn more, see [Accessing and managing read sets with Amazon S3 URIs](#).

## Accessing and managing read sets with Amazon S3 URIs

You can use Amazon S3 URI paths to access your active sequence store read sets. Use Amazon S3 API operations to list, share, and download your read sets. This makes it possible for you to collaborate and share your data because you can share access across users in the data owner's account in the Region, or you can share through creating role users outside of the data owner's account that you can assume. Archived read sets aren't accessible by using Amazon S3 URIs until they have been activated. When a read set is activated, it's restored to the same URI path each time.

With data loaded into HealthOmics stores, because the Amazon S3 URI is based on Amazon S3 access points, you can directly integrate with industry standard tools that read Amazon S3 URIs, such as the following:

- Visual analysis applications such as Integrative Genomics Viewer (IGV) or UCSC Genome Browser.
- Common workflows with Amazon S3 extensions such as CWL, WDL, and NextFlow.
- Any tool that can authenticate and read from access point Amazon S3 URIs or read presigned Amazon S3 URIs.
- Amazon S3 utilities such as Mountpoint or CloudFront.

Amazon S3 Mountpoint makes it possible for you to use an Amazon S3 bucket as a local file system. To learn more about Mountpoint and to install it for use, see [Mountpoint for Amazon S3](#).

Amazon CloudFront is a content delivery network (CDN) service built for high performance, security, and developer convenience. To learn more about using Amazon CloudFront, see [the Amazon CloudFront documentation](#). To set up CloudFront with a sequence store, contact the AWS HealthOmics team.

The data owner root account is enabled for the actions S3:GetObject, S3:GetObjectTagging, and S3:ListBucket on the sequence store prefix. For a user in the account to access the data, you create an IAM policy and attach it to the user or role. For an example policy, see [Permissions for data access using Amazon S3 URIs](#).

You can use the following Amazon S3 API operations on the active read sets to list and retrieve your data. You can access archived read sets through Amazon S3 URIs after they have been activated.

- [GetObject](#)— Retrieves an object from Amazon S3.
- [HeadObject](#)— The HEAD operation retrieves metadata from an object without returning the object itself. This operation is useful if you only want an object's metadata.
- [ListObjects and ListObject v2](#)— Returns some or all (up to 1,000) of the objects in a bucket.
- [CopyObject](#)— Creates a copy of an object that's already stored in Amazon S3. HealthOmics supports copying to an Amazon S3 access point, but not writing to an access point.

HealthOmics sequence stores maintain the semantic identity of files through ETags. Throughout a lifecycle of a file, the Amazon S3 ETag, which is based on bitwise identity, may change, however, the HealthOmics ETag remains the same. To learn more, see [ETag calculation and data provenance](#).

## Amazon S3 URI structure in HealthOmics storage

All files with Amazon S3 URIs have `omics:subjectId` and `omics:sampleId` resource tags. You can use these tags to share access by using IAM policies through a pattern such as `"s3:ExistingObjectTag/omics:subjectId": "pattern desired"`.

The file structure is as follows:

```
.../account_id/sequenceStore/seq_store_id/readSet/read_set_id/files.
```

For files imported into sequence stores from Amazon S3, the sequence store attempts to maintain the original source name. When the names conflict, the system appends read set information to ensure that the file names are unique. For instance, for fastq read sets, if both file names are the same, to make the names unique, `sourceX` is inserted before `.fastq.gz` or `.fq.gz`. For a direct upload, the file names follow the following patterns:

- For FASTQ— `read_set_name_sourceX.fastq.gz`

- For uBAM/BAM/CRAM— *read\_set\_name.file extension* with extensions of .bam or .cram. An example is NA193948.bam.

For read sets that are BAM or CRAM, index files are automatically generated during the ingestion process. For the index files generated, the proper index extension at the end of the file name is applied. It has the pattern *<name of the Source the index is on>.<file index extension>*. The index extensions are .bai or .crai.

## Using Hosted or Local IGV to access read sets

IGV is a genome browser used to analyze BAM and CRAM files. It requires both the file and the index because it only displays a portion of the genome at a time. IGV can be downloaded and used locally, and there are guides to creating an AWS hosted IGV. The public web version isn't supported because it requires CORS.

Local IGV relies on the local AWS configuration to access files. Ensure that the role used in that configuration has a policy attached that enables kms:Decrypt and s3:GetObject permissions to the s3 URI of the read sets being accessed. After that, in IGV, you can use “File > load from URL” and paste in the URI for the source and index. Alternatively, presigned URLs can be generated and used in the same manner, which will bypass the AWS configuration. Note that CORS isn't supported with Amazon S3 URI access, so requests relying on CORS aren't supported.

The example AWS Hosted IGV relies on AWS Cognito to create the correct configurations and permissions inside the environment. Ensure that a policy is created that enables kms:Decrypt and s3:GetObject permissions to the Amazon S3 URI of the read sets being accessed, and add this policy to the role that's assigned to the Cognito user pool. After that, in IGV, you can use “File > load from URL” and enter in the URI for the source and index. Alternatively, presigned URLs can be generated and used in the same manner, which bypasses the AWS configuration.

Note that the sequence store will not appear under the “Amazon” tab because that only displays buckets owned by you in the Region in which the AWS profile is configured.

## Using Samtools or HTSlib in HealthOmics

HTSlib is the core library that's shared by several tools such as Samtools, rSamtools, PySam, and others. Use HTSlib version 1.20 or later to get seamless support for Amazon S3 Access Points. For older versions of the HTSlib library, you can use the following workarounds:



- Set the environment variable for the HTS Amazon S3 host with: `export HTS_S3_HOST="s3.region.amazonaws.com"`.
- Generate a presigned URL for the files that you want to use. If a BAM or CRAM is being used, ensure that a presigned URL is generated for both the file and the index. After that, both files can be used with the libraries.
- Use Mountpoint to mount the sequence store or read set prefix in the same environment where you're using HTSlib libraries. From here, the files can be accessed by using local file paths.

## Using Mountpoint HealthOmics

Mountpoint for Amazon S3 is a simple, high-throughput file client for [mounting an Amazon S3 bucket as a local file system](#). With Mountpoint for Amazon S3, your applications can access objects stored in Amazon S3 through file operations such as open and read. Mountpoint for Amazon S3 automatically translates these operations into Amazon S3 object API calls, giving your applications access to the elastic storage and throughput of Amazon S3 through a file interface.

Mountpoint can be installed by using [the Mountpoint installation instructions](#). Mountpoint uses the AWS Profile that's local to the installation and works at an Amazon S3 prefix level. Ensure that the profile being used has a policy that enables `s3:GetObject`, `s3:ListBucket`, and `kms:Decrypt` permissions to the Amazon S3 URI prefix of the read set(s) or sequence store being accessed. After that, the bucket can be mounted by using the following path:

```
mount-s3 access point arn local path to mount --prefix prefix to sequence store or read set --region region
```

## Using CloudFront with HealthOmics

Amazon CloudFront is a content delivery network (CDN) service that's built for high performance, security, and developer convenience. Customers that want to use CloudFront must work with the Service team to turn on the CloudFront distribution. Work with your account team to engage the HealthOmics service team.

## Activating read sets

You can activate read sets that are archived with the **start-read-set-activation-job** API operation, or through the AWS CLI, as shown in the following example. Replace the *sequence store ID* and *read set id* with your sequence store ID and read set IDs.

```
aws omics start-read-set-activation-job
  --sequence-store-id sequence store ID \
  --sources readSetId=read set ID readSetId=read set id_1 read set id_2
```

You receive a response that contains the activation job information, as shown in the following.

```
{
  "id": "12345678",
  "sequenceStoreId": "1234567890",
  "status": "SUBMITTED",
  "creationTime": "2022-10-22T00:50:54.670000+00:00"
}
```

After the activation job starts, you can monitor its progress with the **get-read-set-activation-job** API operation. The following is an example of how to use the AWS CLI to check your activation job status. Replace *job ID* and *sequence store ID* with your sequence store ID and job IDs, respectively.

```
aws omics get-read-set-activation-job --id job ID --sequence-store-id sequence store ID
```

The response summarizes the activation job, as shown in the following.

```
{
  "id": 123567890,
  "sequenceStoreId": 123467890,
  "status": "SUBMITTED",
  "statusUpdateReason": "The job is submitted and will start soon.",
  "creationTime": "2022-10-22T00:50:54.670000+00:00",
  "sources": [
    {
      "readSetId": <reads set id_1>,
      "status": "NOT_STARTED",
      "statusUpdateReason": "The source is queued for the job."
    },
    {
      "readSetId": <read set id_2>,
      "status": "NOT_STARTED",
      "statusUpdateReason": "The source is queued for the job."
    }
  ]
}
```

```
}

```

You can check the status of an activation job with the **get-read-set-metadata** API operation. Possible statuses are ACTIVE, ACTIVATING, and ARCHIVED. In the following example, replace *sequence store ID* with your sequence store ID, and replace *read set ID* with your read set ID.

```
aws omics get-read-set-metadata --sequence-store-id sequence store ID --id read set ID

```

The following response shows you that the read set is active.

```
{
  "id": "12345678",
  "arn": "arn:aws:omics:us-west-2:555555555555:sequenceStore/1234567890/readSet/12345678",
  "sequenceStoreId": "0123456789",
  "subjectId": "mySubject",
  "sampleId": "mySample",
  "status": "ACTIVE",
  "name": "HG00100",
  "description": "HG00100 aligned to HG38 BAM",
  "fileType": "BAM",
  "creationTime": "2022-07-13T23:25:20Z",
  "sequenceInformation": {
    "totalReadCount": 1513467,
    "totalBaseCount": 163454436,
    "generatedFrom": "Pulled from SRA",
    "alignment": "ALIGNED"
  },
  "referenceArn": "arn:aws:omics:us-west-2:555555555555:referenceStore/0123456789/reference/0000000001",
  "files": {
    "source1": {
      "totalParts": 2,
      "partSize": 10485760,
      "contentLength": 17112283,
      "s3Access": {
        "s3Uri": "s3://accountID-sequence store ID-ajdpi90jdas90a79fh9a8ja98jdfa9jdf98-s3alias/592761533288/sequenceStore/2015356892/readSet/9515444019/import_source1.fastq.gz"
      }
    }
  }
},
}
```

```

    "index": {
      "totalParts": 1,
      "partSize": 53216,
      "contentLength": 10485760
      "s3Access": {
        "s3Uri": "s3://accountID-sequence store ID-ajdpi90jdass90a79fh9a8ja98jdfa9jff98-
s3alias/592761533288/sequenceStore/2015356892/readSet/9515444019/
import_source1.fastq.gz"
      },
    }
  },
  "creationType": "IMPORT",
  "etag": {
    "algorithm": "BAM_MD5up",
    "source1": "d1d65429212d61d115bb19f510d4bd02"
  }
}

```

You can view all read set activation jobs by using **list-read-set-activation-jobs**, as shown in the following example. In the following example, replace *sequence store ID* with your sequence store ID.

```
aws omics list-read-set-activation-jobs --sequence-store-id sequence store ID
```

You receive the following response.

```

{
  "activationJobs": [
    {
      "id": 1234657890,
      "sequenceStoreId": "1234567890",
      "status": "COMPLETED",
      "creationTime": "2022-10-22T01:33:38.079000+00:00",
      "completionTime": "2022-10-22T01:34:28.941000+00:00"
    }
  ]
}

```

# HealthOmics Analytics

HealthOmics Analytics supports the storage and analysis of genomic variants and annotations. Analytics provides two types of storage resources - Variant stores and Annotation stores. You use these resources to store, transform, and query genomic variant data and annotation data. After you import data into a datastore, you can use Athena to perform advanced analytics on the data.

You can use the HealthOmics console or API to create and manage stores, import data, and share analytic store data with collaborators.

Variant stores support data in VCF formats, and annotation stores support TSV/CSV and GFF3 formats. Genomic coordinates are represented as zero-based, half-closed half-open intervals. When your data is in the HealthOmics Analytics data store, access to the VCF files is managed through AWS Lake Formation. You can then query the VCF files by using Amazon Athena. Queries must use Athena query engine version 3. To read more about Athena query engine versions, see the [Amazon Athena documentation](#).

## Topics

- [Creating variant stores](#)
- [Creating variant store import jobs](#)
- [Creating annotation stores](#)
- [Creating annotation store import jobs](#)
- [Deleting analytics stores](#)
- [Querying analytics data](#)
- [Sharing analytics stores](#)

## Creating variant stores

The following topics describe how to create variant stores using the console and the API.

### Topics

- [Creating variant stores using the console](#)
- [Creating variant stores using the API](#)

## Creating variant stores using the console

You can create a variant store using the HealthOmics console.

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Variant stores**.
3. On the **Create variant store** page, provide the following information
  - **Variant store name** - A unique name for this store.
  - **Description** (optional) - A description of this variant store.
  - **Reference genome** - The reference genome for this variant store.
  - **Data Encryption** - Choose whether you want data encryption to be owned and managed by AWS or by yourself.
  - **Tags** (optional) - Provide up to 50 tags for this variant store.
4. Choose **Create variant store**.

## Creating variant stores using the API

You can HealthOmics API operations to create and manage variant stores. You can also perform these operations with the AWS CLI.

The following example uses the AWS CLI to create a variant store.

```
aws omics create-variant-store --name myvariantstore \  
  --reference referenceArn="arn:aws:omics:us-  
west-2:555555555555:referenceStore/123456789/reference/5987565360"
```

To confirm the creation of your variant store, you will receive the following response.

```
{  
  "creationTime": "2022-11-03T18:19:52.296368+00:00",  
  "id": "45aeb91d5678",  
  "name": "myvariantstore",  
  "reference": {  
    "referenceArn": "arn:aws:omics:us-west-2:555555555555:referenceStore/123456789/  
reference/5987565360"  
  },  
  "status": "CREATING"
```

```
}
```

To learn more about a variant store, use the **get-variant-store** API.

```
aws omics get-variant-store --name myvariantstore
```

You will receive the following response.

```
{
  "id": "45aeb91d5678",
  "reference": {
    "referenceArn": "arn:aws:omics:us-west-2:555555555555:referenceStore/123456789/
reference/5987565360"
  },
  "status": "ACTIVE",
  "storeArn": "arn:aws:omics:us-west-2:555555555555:variantStore/myvariantstore",
  "name": "myvariantstore",
  "creationTime": "2022-11-03T18:19:52.296368+00:00",
  "updateTime": "2022-11-03T18:30:56.272792+00:00",
  "tags": {},
  "storeSizeBytes": 0
}
```

To view all variant stores associated with an account, use the **list-variant-stores** API.

```
aws omics list-variant-stores
```

You will receive a response that lists all variant stores, along with their IDs, statuses, and other details, as shown in the following example response.

```
{
  "variantStores": [
    {
      "id": "45aeb91d5678",
      "reference": {
        "referenceArn": "arn:aws:omics:us-
west-2:555555555555:referenceStore/5506874698"
      },
      "status": "ACTIVE",
      "storeArn": "arn:aws:omics:us-west-2:555555555555:variantStore/
new_variant_store",
    }
  ]
}
```

```

        "name": "variantstore",
        "creationTime": "2022-11-03T18:19:52.296368+00:00",
        "updateTime": "2022-11-03T18:30:56.272792+00:00",
        "statusMessage": "",
        "storeSizeBytes": 141526
    }
]
}

```

You can also filter the responses for the **list-variant-stores** API based on statuses or other criteria.

VCF Files imported into analytic stores created on or after May 15, 2023 have defined schemas for Variant Effect Predictor (VEP) annotations. This makes it easier to query and parse imported VCF data. The change does not impact stores created before May 15, 2023, except if the `annotation fields` parameter is included in the API or CLI call. For these stores, using the `annotation fields` parameter will cause the request to fail.

## Creating variant store import jobs

The following example shows how to use the AWS CLI to create an import job for a variant store.

```

aws omics start-variant-import-job \
  --destination-name myvariantstore \
  --runLeftNormalization false \
  --role-arn arn:aws:iam::555555555555:role/roleName \
  --items source=s3://my-omics-bucket/sample.vcf.gz source=s3://my-omics-bucket/
sample2.vcf.gz

```

```

{
  "destinationName": "store_a",
  "roleArn": "...",
  "runLeftNormalization": false,
  "items": [
    {"source": "s3://my-omics-bucket/sample.vcf.gz"},
    {"source": "s3://my-omics-bucket/sample2.vcf.gz"}
  ]
}

```

For stores created after May 15, 2023, the following example shows how to add the `--annotation-fields` parameter. The annotation fields are defined with the `import`.



```
aws omics start-variant-import-job \
  --destination-name annotationparsingvariantstore \
  --role-arn arn:aws:iam::123456789012:role/<role_name> \
  --items source=s3://pathToS3/sample.vcf
  --annotation-fields '{"VEP": "CSQ"}'
```

```
{
  "jobId": "981e2286-e954-4391-8a97-09aefc343861"
}
```

Use **get-variant-import-job** to check the status.

```
aws omics get-variant-import-job --job-id 08279950-a9e3-4cc3-9a3c-a574f9c9e229
```

You'll receive a JSON response that shows the status of your import job. VEP annotations in the VCF are parsed for information stored in the INFO column as an ID/Value pair. The default ID for [Ensembl Variant Effect Predictor](#) annotations INFO column is CSQ, but you can use the `--annotation-fields` parameter to indicate a custom value used in the INFO column. Parsing is currently supported for VEP annotations.

For a store created before May 15, 2023 or for VCF files that don't include VEP annotation, the response doesn't include any annotation fields.

```
{
  "creationTime": "2023-04-11T17:52:37.241958+00:00",
  "destinationName": "annotationparsingvariantstore",
  "id": "7a1c67e3-b7f9-434d-817b-9c571fd63bea",
  "items": [
    {
      "jobStatus": "COMPLETED",
      "source": "s3://DOC-EXAMPLE-BUCKET/NA12878.2k.garvan.vcf"
    }
  ],
  "roleArn": "arn:aws:iam::555555555555:role/<role_name>",
  "runLeftNormalization": false,
  "status": "COMPLETED",
  "updateTime": "2023-04-11T17:58:22.676043+00:00",
}
```

The VEP annotations that are a part of VCF files are stored as predefined schema with the following structure. The extras field can be used to store any additional VEP fields that aren't included in the default schema.

```
annotations struct<
  vep: array<struct<
    allele:string,
    consequence: array<string>,
    impact:string,
    symbol:string,
    gene:string,
    `feature_type`: string,
    feature: string,
    biotype: string,
    exon: struct<rank:string, total:string>,
    intron: struct<rank:string, total:string>,
    hgpsc: string,
    hgvsp: string,
    `cdna_position`: string,
    `cds_position`: string,
    `protein_position`: string,
    `amino_acids`: struct<reference:string, variant: string>,
    codons: struct<reference:string, variant: string>,
    `existing_variation`: array<string>,
    distance: string,
    strand: string,
    flags: array<string>,
    symbol_source: string,
    hgnc_id: string,
    `extras`: map<string, string>
  >>
>
```

The parsing is performed with a best effort approach. If the VEP entry doesn't follow the [VEP standard specifications](#), it won't be parsed and the row in the array will be empty.

For a new variant store, the response for **get-variant-import-job** would include the annotation fields, as shown.

```
aws omics get-variant-import-job --job-id 08279950-a9e3-4cc3-9a3c-a574f9c9e229
```

You'll receive a JSON response that shows the status of your import job.

```
{
  "creationTime": "2023-04-11T17:52:37.241958+00:00",
  "destinationName": "annotationparsingvariantstore",
  "id": "7a1c67e3-b7f9-434d-817b-9c571fd63bea",
  "items": [
    {
      "jobStatus": "COMPLETED",
      "source": "s3://DOC-EXAMPLE-BUCKET/NA12878.2k.garvan.vcf"
    }
  ],
  "roleArn": "arn:aws:iam::123456789012:role/<role_name>",
  "runLeftNormalization": false,
  "status": "COMPLETED",
  "updateTime": "2023-04-11T17:58:22.676043+00:00",
  "annotationFields" : {"VEP": "CSQ"}
}
```

You can use **list-variant-import-jobs** to see all import jobs and their statuses.

```
aws omics list-variant-import-jobs --ids 7a1c67e3-b7f9-434d-817b-9c571fd63bea
```

The response returned will have information as follows.

```
{
  "variantImportJobs": [
    {
      "creationTime": "2023-04-11T17:52:37.241958+00:00",
      "destinationName": "annotationparsingvariantstore",
      "id": "7a1c67e3-b7f9-434d-817b-9c571fd63bea",
      "roleArn": "arn:aws:iam::555555555555:role/roleName",
      "runLeftNormalization": false,
      "status": "COMPLETED",
      "updateTime": "2023-04-11T17:58:22.676043+00:00",
      "annotationFields" : {"VEP": "CSQ"}
    }
  ]
}
```

If necessary, you can cancel an import job with the following command.

```
aws omics cancel-variant-import-job
  --job-id edd7b8ce-xmpl-47e2-bc99-258cac95a508
```

## Creating annotation stores

An annotation store is a data store representing an annotation database, such as one from a TSV, VCF, or GFF file. If the same reference genome is specified, annotation stores are mapped to the same coordinate system as variant stores during an import. The following topics show how to use the HealthOmics console and AWS CLI to create and manage annotation stores.

### Topics

- [Create an annotation store using the console](#)
- [Create an annotation store using the API](#)
- [Creating new versions of annotation stores](#)

## Create an annotation store using the console

The following procedure to create annotation stores using the HealthOmics console.

### To create an annotation store

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Annotation stores**.
3. On the **Annotation stores** page, choose **Create annotation store**.
4. On the **Create annotation store** page, provide the following information
  - **Annotation store name** - A unique name for this store.
  - **Description** (optional) - A description of this reference genome.
  - **Data format and schema details** - Select data file format and upload the schema definition for this store.
  - **Reference genome** - The reference genome for this annotation.
  - **Data Encryption** - Choose whether you want data encryption to be owned and managed by AWS or by yourself.

- **Tags (optional)** - Provide up to 50 tags for this annotation store.

## 5. Choose **Create annotation store**.

### Create an annotation store using the API

The following example shows how to create an annotation store using the AWS CLI. For all AWS CLI and API operations, you must specify the format of your data.

```
aws omics create-annotation-store --name my_annotation_store \  
  --store-format VCF \  
  --reference referenceArn="arn:aws:omics:us-  
west-2:555555555555:referenceStore/6505293348/reference/5987565360"  
  --version-name new_version
```

You receive the following response to confirm the creation of your annotation store.

```
{  
  "creationTime": "2022-08-24T20:34:19.229500Z",  
  "id": "3b93cdef69d2",  
  "name": "my_annotation_store",  
  "reference": {  
    "referenceArn": "arn:aws:omics:us-  
west-2:555555555555:referenceStore/6505293348/reference/5987565360"  
  },  
  "status": "CREATING"  
  "versionName": "my_version"  
}
```

To learn more about an annotation store, use the **get-annotation-store** API.

```
aws omics get-annotation-store --name my_annotation_store
```

You receive the following response.

```
{  
  "id": "eeb019ac79c2",  
  "reference": {  
    "referenceArn": "arn:aws:omics:us-  
west-2:555555555555:referenceStore/5638433913/reference/5871590330"  
  },  
}
```

```
"status": "ACTIVE",
"storeArn": "arn:aws:omics:us-west-2:555555555555:annotationStore/gffstore",
"name": "my_annotation_store",
"creationTime": "2022-11-05T00:05:19.136131+00:00",
"updateTime": "2022-11-05T00:10:36.944839+00:00",
"tags": {},
"storeFormat": "GFF",
"statusMessage": "",
"storeSizeBytes": 0,
"numVersions": 1
}
```

To view all annotation stores associated with an account, use the **list-annotation-stores** API operation.

```
aws omics list-annotation-stores
```

You receive a response that lists all annotation stores, along with their IDs, statuses, and other details, as shown in the following example response.

```
{
  "annotationStores": [
    {
      "id": "4d8f3eada259",
      "reference": {
        "referenceArn": "arn:aws:omics:us-
west-2:555555555555:referenceStore/5638433913/reference/5871590330"
      },
      "status": "CREATING",
      "name": "gffstore",
      "creationTime": "2022-09-27T17:30:52.182990+00:00",
      "updateTime": "2022-09-27T17:30:53.025362+00:00"
    }
  ]
}
```

You can also filter responses based on status or other criteria.

## Creating new versions of annotation stores

You can create new versions of annotation stores to collect different versions of your annotation databases. This helps you organize your annotation data, which is updated regularly.

To create a new version of an existing annotation store, use the **create-annotation-store-version** API as shown in the following example.

```
aws omics create-annotation-store-version \  
  --name my_annotation_store \  
  --version-name my_version
```

You will get the following response with the annotation store version ID, confirming that a new version of your annotation has been created.

```
{  
  "creationTime": "2023-07-21T17:15:49.251040+00:00",  
  "id": "3b93cdef69d2",  
  "name": "my_annotation_store",  
  "reference": {  
    "referenceArn": "arn:aws:omics:us-  
west-2:555555555555:referenceStore/6505293348/reference/5987565360"  
  },  
  "status": "CREATING",  
  "versionName": "my_version"  
}
```

To update the description of an annotation store version, you can use **update-annotation-store-version** to add updates to an annotation store version.

```
aws omics update-annotation-store-version \  
  --name my_annotation_store \  
  --version-name my_version \  
  --description "New Description"
```

You will receive the following response, confirming that the annotation store version has been updated.

```
{  
  "storeId": "4934045d1c6d",  
  "id": "2a3f4a44aa7b",  
  "description": "New Description",  
  "status": "ACTIVE",  
  "name": "my_annotation_store",  
  "versionName": "my_version",  
  "creation Time": "2023-07-21T17:20:59.380043+00:00",
```

```
    "updateTime": "2023-07-21T17:26:17.892034+00:00"  
  }
```

To view the details of an annotation store version, use **get-annotation-store-version**.

```
aws omics get-annotation-store-version --name my_annotation_store --version-name  
my_version
```

You will receive a response with the version name, status, and other details.

```
{  
  "storeId": "4934045d1c6d",  
  "id": "2a3f4a44aa7b",  
  "status": "ACTIVE",  
  "versionArn": "arn:aws:omics:us-west-2:555555555555:annotationStore/  
my_annotation_store/version/my_version",  
  "name": "my_annotation_store",  
  "versionName": "my_version",  
  "creationTime": "2023-07-21T17:15:49.251040+00:00",  
  "updateTime": "2023-07-21T17:15:56.434223+00:00",  
  "statusMessage": "",  
  "versionSizeBytes": 0  
}
```

To view all versions of an annotation store, you can use **list-annotation-store-versions**, as shown in the following example.

```
aws omics list-annotation-store-versions --name my_annotation_store
```

You will receive a response with the following information

```
{  
  "annotationStoreVersions": [  
    {  
      "storeId": "4934045d1c6d",  
      "id": "2a3f4a44aa7b",  
      "status": "CREATING",  
      "versionArn": "arn:aws:omics:us-west-2:555555555555:annotationStore/  
my_annotation_store/version/my_version_2",  
      "name": "my_annotation_store",  
    }  
  ]  
}
```



```
    "versionName": "my_version_2",
    "creationTime": "2023-07-21T17:20:59.380043+00:00",
    "versionSizeBytes": 0
  },
  {
    "storeId": "4934045d1c6d",
    "id": "4934045d1c6d",
    "status": "ACTIVE",
    "versionArn": "arn:aws:omics:us-west-2:555555555555:annotationStore/
my_annotation_store/version/my_version_1",
    "name": "my_annotation_store",
    "versionName": "my_version_1",
    "creationTime": "2023-07-21T17:15:49.251040+00:00",
    "updateTime": "2023-07-21T17:15:56.434223+00:00",
    "statusMessage": "",
    "versionSizeBytes": 0
  }
}
```

If you no longer need an annotation store version, you can use **delete-annotation-store-versions** to delete an annotation store version, as shown in the following example.

```
aws omics delete-annotation-store-versions --name my_annotation_store --versions
my_version
```

If the store version is deleted without errors, you will receive the following response.

```
{
  "errors": []
}
```

If there are errors, you will receive a response with the details of the errors, as shown.

```
{
  "errors": [
    {
      "versionName": "my_version",
      "message": "Version with versionName: my_version was not found."
    }
  ]
}
```

If you try to delete an annotation store version that has an active import job, you will receive a response with an error, as shown.

```
{
  "errors": [
    {
      "versionName": "my_version",
      "message": "version has an inflight import running"
    }
  ]
}
```

In this case, you can force deletion of the annotation store version, as shown in the following example.

```
aws omics delete-annotation-store-versions --name my_annotation_store --versions
my_version --force
```

## Creating annotation store import jobs

### Topics

- [Create an annotation import job using the API](#)
- [Additional parameters for TSV and VCF formats](#)
- [Creating TSV formatted annotation stores](#)
- [Starting VCF formatted import jobs](#)

## Create an annotation import job using the API

The following example shows how to use the AWS CLI to start an annotation import job.

```
aws omics start-annotation-import-job \
  --destination-name myannostore \
  --version-name myannostore \
  --role-arn arn:aws:iam::123456789012:role/roleName \
  --items source=s3://my-omics-bucket/sample.vcf.gz
  --annotation-fields '{"VEP": "CSQ"}
```

Annotation stores created before May 15, 2023 return an error message if the **annotation-fields** is included. They don't return output for any API operations involved with annotation store import jobs.

You can then use the **get-annotation-import-job** API operation and the `job ID` parameter to learn more details about the annotation import job.

```
aws omics get-annotation-import-job --job-id 9e4198fb-fa85-446c-9301-9b823a1a8ba8
```

You receive the following response, including the annotation fields.

```
{
  "creationTime": "2023-04-11T19:09:25.049767+00:00",
  "destinationName": "parsingannotationstore",
  "versionName": "parsingannotationstore",
  "id": "9e4198fb-fa85-446c-9301-9b823a1a8ba8",
  "items": [
    {
      "jobStatus": "COMPLETED",
      "source": "s3://my-omics-bucket/sample.vcf"
    }
  ],
  "roleArn": "arn:aws:iam::555555555555:role/roleName",
  "runLeftNormalization": false,
  "status": "COMPLETED",
  "updateTime": "2023-04-11T19:13:09.110130+00:00",
  "annotationFields" : {"VEP": "CSQ"}
}
```

To view all annotation store import jobs, use **list-annotation-import-jobs** .

```
aws omics list-annotation-import-jobs --ids 9e4198fb-fa85-446c-9301-9b823a1a8ba8
```

The response includes the details and statuses of your annotation store import jobs.

```
{
  "annotationImportJobs": [
    {
      "creationTime": "2023-04-11T19:09:25.049767+00:00",
```

```

    "destinationName": "parsingannotationstore",
    "versionName": "parsingannotationstore",
    "id": "9e4198fb-fa85-446c-9301-9b823a1a8ba8",
    "roleArn": "arn:aws:iam::555555555555:role/roleName",
    "runLeftNormalization": false,
    "status": "COMPLETED",
    "updateTime": "2023-04-11T19:13:09.110130+00:00",
    "annotationFields" : {"VEP": "CSQ"}
  }
]
}

```

## Additional parameters for TSV and VCF formats

For TSV and VCF formats, there are additional parameters that inform the API of how to parse your input.

### Important

CSV annotation data that's exported with query engines directly returns information from the dataset import. If the imported data contains formulas or commands, the file might be subject to CSV injection. Therefore, files exported with query engines can prompt security warnings. To avoid malicious activity, turn off links and macros when reading export files.

The TSV parser also performs basic bioinformatics operations, like left normalization and standardization of genomics coordinates, that are listed in the following table.

Format type	Description
Generic	Generic text file. No genomic information.
CHR_POS	Start position - 1, Add end position, which is the same as POS.
CHR_POS_REF_ALT	Contains contig, 1-base position, ref and alt allele information.
CHR_START_END_REF_ALT_ONE_BASE	Contains contig, start, end, ref and alt allele information. Coordinates are 1-based.

Format type	Description
CHR_START_END_ZERO_BASE	Contains contig, start, and end positions. Coordinates are 0-based.
CHR_START_END_ONE_BASE	Contains contig, start, and end positions. Coordinates are 1-based.
CHR_START_END_REF_ALT_ZERO_BASE	Contains contig, start, end, ref and alt allele information. Coordinates are 0-based.

A TSV import annotation store request looks like the following example.

```
aws omics start-annotation-import-job \
--destination-name tsv_anno_example \
--role-arn arn:aws:iam::555555555555:role/demoRole \
--items source=s3://demodata/genomic_data.bed.gz \
--format-options '{ "tsvOptions": {
    "readOptions": {
      "header": false,
      "sep": "\t"
    }
  }
}'
```

## Creating TSV formatted annotation stores

The following example creates an annotation store using a tab limited file that contains a header, rows, and comments. The coordinates are CHR\_START\_END\_ONE\_BASED, and it contains the HG19 gene map from the [OMIM's Synopsis of the Human Gene Map](#).

```
aws omics create-annotation-store --name mimgenemap \
--store-format TSV \
--reference=referenceArn=arn:aws:omics:us-
west-2:555555555555:referenceStore/6505293348/reference/2310864158 \
--store-options=tsvStoreOptions='{
  annotationType=CHR_START_END_ONE_BASE,
  formatToHeader={CHR=chromosome, START=genomic_position_start,
  END=genomic_position_end},
```

```

schema=[
  {chromosome=STRING},
  {genomic_position_start=LONG},
  {genomic_position_end=LONG},
  {cyto_location=STRING},
  {computed_cyto_location=STRING},
  {mim_number=STRING},
  {gene_symbols=STRING},
  {gene_name=STRING},
  {approved_gene_name=STRING},
  {entrez_gene_id=STRING},
  {ensembl_gene_id=STRING},
  {comments=STRING},
  {phenotypes=STRING},
  {mouse_gene_symbol=STRING}]]'

```

You can import files with or without a header. To indicate this in a CLI request, use `header=false`, as shown in the following import job example.

```

aws omics start-annotation-import-job \
  --role-arn arn:aws:iam::555555555555:role/demoRole \
  --items=source=s3://DOC-EXAMPLE-BUCKET/annotation-examples/hg38_genemap2.txt \
  --destination-name output-bucket \
  --format-options=tsvOptions='{readOptions={sep="\t",header=false,comment="#"}}'

```

The following example creates an annotation store for a bed file. A bed file is a simple tab delimited file. In this example, the columns are chromosome, start, end, and region name. The coordinates are zero-based, and the data does not have a header.

```

aws omics create-annotation-store \
  --name cexbed --store-format TSV \
  --reference=referenceArn=arn:aws:omics:us-
west-2:555555555555:referenceStore/6505293348/reference/2310864158 \
  --store-options=tsvStoreOptions='{
annotationType=CHR_START_END_ZERO_BASE,
formatToHeader={CHR=chromosome, START=start, END=end},
schema=[{chromosome=STRING}, {start=LONG}, {end=LONG}, {name=STRING}]}'

```

You can then import the bed file into the annotation store by using the following the CLI command.

```

aws omics start-annotation-import-job \

```

```
--role-arn arn:aws:iam::555555555555:role/demoRole \
--items=source=s3://DOC-EXAMPLE-BUCKET/TruSeq_Exome_TargetedRegions_v1.2.bed \
--destination-name cexbed \
--format-options=tsvOptions='{readOptions={sep="\t",header=false,comment="#"}}'
```

The following example creates an annotation store for a tab delimited file that contains the first few columns of a VCF file, followed by columns with annotation information. It contains genome positions with information on the chromosome, start, reference and alternate alleles, and it contains a header.

```
aws omics create-annotation-store --name gnomadchrX --store-format TSV \
--reference=referenceArn=arn:aws:omics:us-
west-2:555555555555:referenceStore/6505293348/reference/2310864158 \
--store-options=tsvStoreOptions='{
  annotationType=CHR_POS_REF_ALT,
  formatToHeader={CHR=chromosome, POS=start, REF=ref, ALT=alt},
  schema=[
    {chromosome=STRING},
    {start=LONG},
    {ref=STRING},
    {alt=STRING},
    {filters=STRING},
    {ac_hom=STRING},
    {ac_het=STRING},
    {af_hom=STRING},
    {af_het=STRING},
    {an=STRING},
    {max_observed_heteroplasmy=STRING}]]'
```

You would then import the file into the annotation store using the following the CLI command.

```
aws omics start-annotation-import-job \
--role-arn arn:aws:iam::555555555555:role/demoRole \
--items=source=s3://DOC-EXAMPLE-BUCKET/
gnomad.genomes.v3.1.sites.chrM.reduced_annotations.tsv \
--destination-name gnomadchrX \
--format-options=tsvOptions='{readOptions={sep="\t",header=true,comment="#"}}'
```

The following example shows how a customer can create an annotation store for a mim2gene file. A mim2gene file provides the links between the genes in OMIM and another gene identifier. It's tab delimited and contains comments.

```
aws omics create-annotation-store \  
  --name mim2gene \  
  --store-format TSV \  
  --reference=referenceArn=arn:aws:omics:us-  
west-2:555555555555:referenceStore/6505293348/reference/2310864158 \  
  --store-options=tsvStoreOptions='  
  {annotationType=GENERIC,  
  formatToHeader={},  
  schema=[  
    {mim_gene_id=STRING},  
    {mim_type=STRING},  
    {entrez_id=STRING},  
    {hgnc=STRING},  
    {ensembl=STRING}]}'
```

You can then import data into your store as follows.

```
aws omics start-annotation-import-job \  
  --role-arn arn:aws:iam::555555555555:role/demoRole \  
  --items=source=s3://xquek-dev-aws/annotation-examples/mim2gene.txt \  
  --destination-name mim2gene \  
  --format-options=tsvOptions='{readOptions={sep="\t",header=false,comment="#"}}'
```

## Starting VCF formatted import jobs

For VCF files, there are two additional inputs, `ignoreQualField` and `ignoreFilterField`, that ignore or include those parameters as shown.

```
aws omics start-annotation-import-job --destination-name annotation_example\  
  --role-arn arn:aws:iam::555555555555:role/demoRole \  
  --items source=s3://demodata/example.garvan.vcf \  
  --format-options '{ "vcfOptions": {  
    "ignoreQualField": false,  
    "ignoreFilterField": false  
  }  
}'
```



You can also cancel an annotation store import, as shown. If the cancellation succeeds, you don't receive a response to this AWS CLI call. However, if the import job ID isn't found or the import job is completed, you receive an error message.

```
aws omics cancel-annotation-import-job --job-id edd7b8ce-xmpl-47e2-bc99-258cac95a508
```

### Note

Your metadata import job history for **get-annotation-import-job**, **get-variant-import-job**, **list-annotation-import-jobs**, and **list-variant-import-jobs** is auto-deleted after two years. The variant and annotation data that's imported isn't auto-deleted and remains in your data stores.

## Deleting analytics stores

When you delete a variant or annotation store, the system also deletes all imported data in that store and any associated tags.

The following example shows how to delete a variant store using the AWS CLI. If the action is successful, the variant store status transitions to DELETING.

```
aws omics delete-variant-store --id <variant-store-id>
```

The following example shows how to delete an annotation store. If the action is successful, the annotation store status transitions to DELETING. Annotation stores can't be deleted if more than one version exists.

```
aws omics delete-annotation-store --id <annotation-store-id>
```

## Querying analytics data

You can perform queries on your variant stores using AWS Lake Formation and Amazon Athena or Amazon EMR. Before you run any queries, complete the setup procedures (described in the following sections) for Lake Formation and Amazon Athena.

For information about Amazon EMR, see [Tutorial: Getting started with Amazon EMR](#)

## Topics

- [Setting up the Lake Formation console](#)
- [Configuring Athena for queries](#)
- [Running queries on variant stores](#)

## Setting up the Lake Formation console

Before you use Lake Formation to manage HealthOmics data stores, perform the following Lake Formation setup procedures.

### Topics

- [Create or verify Lake Formation administrators](#)
- [Create resource links using the Lake Formation console](#)
- [Configure permissions for AWS RAM resource shares](#)

## Create or verify Lake Formation administrators

Before you can create a data lake in Lake Formation, you define one or more administrators.

Administrators are users and roles with permissions to create resource links. You set up data lake administrators per account per region.

### Create an admin user in the Lake Formation console

1. Open the AWS Lake Formation console: [Lake Formation console](#)
2. If the console displays the **Welcome to Lake Formation** panel, choose **Get started**.

Lake Formation adds you to the **Data lake administrators** table.

3. Otherwise, from the left menu, choose **Administrative roles and tasks**.
4. Add any additional administrators as required.

## Create resource links using the Lake Formation console

To make a shared resource that users can query, the default access controls must be disabled. To learn more about disabling default access controls, see [Changing the default security settings for](#)

[your data lake](#) in the Lake Formation documentation. You can create resource links individually or as a group, so that you can access data in Amazon Athena or other AWS services (such as Amazon EMR).

## Creating resource links in the AWS Lake Formation console and sharing them with HealthOmics Analytics users

1. Open the AWS Lake Formation console: [Lake Formation console](#)
2. In the primary navigation bar, choose **Databases**.
3. In the **Databases** table, choose the desired database.
4. From the **Actions** menu, choose **Create resource link**.
5. Enter a **Resource link name**. If you plan to access the database from Athena, enter a name using only lowercase letters (up to 256 characters).
6. Choose **Create**.
7. The new resource link is now listed under **Databases**.

## Grant access to the shared resource using the Lake Formation console

A Lake Formation database administrator can grant access to the shared resource using the following procedure.

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, select the resource link you previously created.
4. From the **Actions** menu, choose **Grant on target**.
5. On the **Grant data permissions** page under **Principals**, choose **IAM users or roles**.
6. From the **IAM users or roles** drop-down menu, find the user to which you want to grant access.
7. Next, under **LF-Tags or catalog resources** card, select the **Named data catalog resources** option.
8. From the **Tables-optional** drop-down menu, select **All Tables** or the table that you previously created.
9. In the **Table permissions** card, under **Table permissions** choose **Describe** and **Select**.
10. Next, choose **Grant**.

To view the Lake Formation permissions, choose **Data lake permissions** from the primary navigation pane. The table shows the available databases and resource links.

## Configure permissions for AWS RAM resource shares

In the AWS Lake Formation console, view the permissions by choosing **Data lake permissions** in the primary navigation bar. On the **Data permissions** page, you can view a table that shows the **Resource types**, **Databases**, and **ARN** that's related to a shared resource under **RAM Resource Share**. If you need to accept an AWS Resource Access Manager (AWS RAM) resource share, AWS Lake Formation notifies you in the console.

HealthOmics can implicitly accept the AWS RAM resource shares during store creation. To accept the AWS RAM resource share, the IAM user or role that calls the `CreateVariantStore` or `CreateAnnotationStore` API operations must allow the following actions:

- `ram:GetResourceShareInvitations` - This action allows HealthOmics to find the invitations.
- `ram:AcceptResourceShareInvitation` - This action allows HealthOmics to accept the invitation by using an FAS token.

Without these permissions, you see an authorization error during store creation.

Here is a sample policy that includes these actions. Add this policy to the IAM user or role that accepts the AWS RAM resource share.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "omics:*",
        "ram:AcceptResourceShareInvitation",
        "ram:GetResourceShareInvitations"
      ],
      "Resource": "*"
    }
  ]
}
```

## Configuring Athena for queries

You can use Athena to query variants and annotations. Before you run any queries, perform the following setup tasks:

### Topics

- [Configure a query results location using the Athena console](#)
- [Configure a workgroup with Athena engine v3](#)

### Configure a query results location using the Athena console

To configure a query results location, follow these steps.

1. Open the Athena console: [Athena console](#)
2. In the primary navigation bar, choose **Query editor**.
3. In the query editor, choose the **Settings** tab, then choose **Manage**.
4. Enter an S3 prefix of a location to save the query result.

### Configure a workgroup with Athena engine v3

To configure a workgroup, follow these steps.

1. Open the Athena console: [Athena console](#)
2. In the primary navigation bar, choose **Workgroups**, then **Create workgroup**.
3. Enter a name for the workgroup.
4. Select **Athena SQL** as the type of engine.
5. Under **Upgrade query engine**, select **Manual**.
6. Under **Query version engine**, select **Athena version 3**.
7. Choose **Create workgroup**.

## Running queries on variant stores

You can perform queries on your variant store using Amazon Athena. Note that genomic coordinates in variant and annotation stores are represented as zero-based, half-closed half-open intervals.

## Run a simple query using the Athena console

The following example shows how to run a simple query.

1. Open the Athena Query editor: [Athena Query editor](#)
2. Under **Workgroup**, select the workgroup that you created during setup.
3. Verify that **Data source** is **AwsDataCatalog**.
4. For **Database**, select the database resource link that you created during the Lake Formation setup.
5. Copy the following query into the **Query Editor** under the **Query 1** tab:

```
SELECT * from omicsvariants limit 10
```

6. Choose **Run** to run the query. The console populates the results table with the first 10 rows of the **omicsvariants** table.

## Run a complex query using the Athena console

The following example shows how to run a complex query. To run this query, import ClinVar into the annotation store.

### Run a complex query

1. Open the Athena Query editor: [Athena Query editor](#)
2. Under **Workgroup**, select the workgroup that you created during setup.
3. Verify that **Data source** is **AwsDataCatalog**.
4. For **Database**, select the database resource link that you created during the Lake Formation setup.
5. Choose the **+** at the top right to create a new query tab named **Query 2**.
6. Copy the following query into the **Query Editor** under the **Query 2** tab:

```
SELECT variants.sampleid,  
       variants.contigname,  
       variants.start,  
       variants."end",  
       variants.referenceallele,  
       variants.alternatealleles,
```

```
variants.attributes AS variant_attributes,  
clinvar.attributes AS clinvar_attributes  
FROM omicsvariants as variants  
INNER JOIN omicsannotations as clinvar ON  
variants.contigname=CONCAT('chr',clinvar.contigname)  
AND variants.start=clinvar.start  
AND variants."end"=clinvar."end"  
AND variants.referenceallele=clinvar.referenceallele  
AND variants.alternatealleles=clinvar.alternatealleles  
WHERE clinvar.attributes['CLNSIG']='Likely_pathogenic'
```

7. Choose **Run** to start running the query.

## Sharing analytics stores

As the owner of a variant store or an annotation store, you can share the store with other AWS accounts. The owner can revoke access to the shared resource by deleting the share.

As the subscriber to a shared store, you first accept the share. You can then define workflows that use the shared store. The data shows up as a table in both AWS Glue and Lake Formation.

When you no longer need access to the store, you delete the share.

See [Cross-account resource sharing in AWS HealthOmics](#) for additional information about resource sharing.

## Creating a store share

To create a store share, use the **create-share** API operation. The principal subscriber is the AWS account of the user who will subscribe to the share. The following example creates a share for a variant store. To share a store with more than one account, you create multiple shares of the same store.

```
aws omics create-share \  
  --resource-arn "arn:aws:omics:us-west-2:555555555555:variantStore/  
omics_dev_var_store" \  
  --principal-subscriber "123456789012" \  
  --name "my_Share-123"
```

If the create is successful, you receive a response with the share ID and status.

```
{
  "shareId": "495c21bedc889d07d0ab69d710a6841e-dd75ab7a1a9c384fa848b5bd8e5a7e0a",
  "name": "my_Share-123",
  "status": "PENDING"
}
```

The share remains in pending state until the subscriber accepts it using the `accept-share` API operation.



# HealthOmics Workflows

With HealthOmics Workflows, you can process and analyze your genomics data using either Ready2Run workflows or private workflows.

*Ready2Run workflows* are created by third-party publishers. They're ready to run without further definition or configuration.

*Private workflows* are workflows that you create. You define the workflow tasks using CWL, WDL, or Nextflow. You add private workflows to a run group to control compute usage. You can share your private workflows with other AWS accounts.

A *run* is a single invocation of a workflow, and a *task* is a single process within the run.

A *run group* is a collection of private workflow runs that share a set of resource limits, such as maximum concurrent runs and maximum run duration. You set these limits to control the compute resources that the run group consumes.

For full examples of how to use HealthOmics workflows, see [HealthOmics Github tutorials](#) or [the AWS workshop end to end tutorial for HealthOmics](#).

## Topics

- [Ready2Run workflows](#)
- [Private workflows](#)
- [Running workflows](#)
- [Using the CloudWatch Logs for troubleshooting](#)

## Ready2Run workflows

Ready2Run workflows are preconfigured workflows published by third-party publishers. Some publishers, such as Sentieon Inc, offer subscription-based workflows. Other Ready2Run workflows don't require a subscription, and some workflows are open source, such as the nf-core workflows.

Ready2Run workflows are well-suited to the following scenarios:

- You want to focus on the analysis of pipeline output and generating results, without the need to set up the underlying infrastructure.
- You want to replicate your results using established workflows.

- As a software developer, you want to integrate your application directly with the HealthOmics SDK.

When you use a Ready2Run workflow, your workflow is preconfigured and can't be edited. In contrast to private workflows, Ready2Run workflows don't support the following:

- Changing the compute resources, run storage, or input file size limits.
- Changing the workflow definition or containers.
- Adding runs to run groups.
- Sharing the workflow.

All Ready2Run workflows provide logs, including CloudWatch logs, that you can use for troubleshooting.

## Topics

- [Using Ready2Run workflows \(console\)](#)
- [Using Ready2Run workflows \(API\)](#)

## Using Ready2Run workflows (console)

Using Ready2Run workflows in the console is similar to using a private workflow. One key difference is that the workflow publisher provides sample data, so that you can try out the workflow without creating your own data.

### To use a Ready2Run workflow in the console

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Ready2Run workflows**.
3. On the **Ready2Run workflows** page, choose the workflow that you want to use. The console opens the details page for that workflow.
4. The details tab lists information such as the name, list price per run, description, workflow language type, run storage capacity, status, creation date, and parameters with descriptions. The details tab also tells you whether the workflow requires a subscription.
5. To use the workflow, choose **Create run**
6. In the **Specify run details** page, enter a run name. You can also add run priority to your run.

7. Enter or select an Amazon S3 location for the workflow run output.
8. For **Run meta data retention mode**, choose whether to retain or remove run meta data.
9. In the **Service role** panel, choose whether to use an existing service role or create a new one.
10. (Optional) Add tags to help identify and manage your run.
11. Choose **Next**.
12. From the **Add parameters** page, choose one of the options to add the run parameter values:
  - Select a parameter file (in JSON format) from an Amazon S3 location.
  - Select a parameter file (in JSON format) from your local drive.
  - Manually enter the parameter values.
  - Run workflow with Ready2Run sample data provided by the workflow publisher.
13. If you upload a JSON file, the console parses the file and performs inline validation. You can then manually update the values of your parameters as needed.
14. Choose **Next**.
15. Review your inputs, then choose **Start run**.

## Using Ready2Run workflows (API)

Most of the API operations for creating runs and workflows behave similarly for both Ready2Run and private workflows.

To return a list of available Ready2Run workflows, use **list-workflows** with the `type` parameter set to `READY2RUN`.

```
aws omics list-workflows --type READY2RUN
```

After you identify the workflow to run from the **list-workflows** response, you can use **get-workflow** with the `--id` parameter to get more details.

```
aws omics get-workflow --type READY2RUN --id workflow id
```

To run a Ready2Run workflow, you can use **start-run** API operation with the `workflow-type` parameter set to `READY2RUN`, as shown in the following example

```
aws-omics start-run \
```

```
--workflow-type READY2RUN \  
--workflow-id workflow id \  
--output-uri &example-s3-bucket; \  
--role-arn arn:aws:iam::1234567892012:role/service-role/OmicsWorkflow-20221004T164236 \  
\  
--parameters file:///path/to/parameters.json
```

To monitor your run, you can use the **get-run** API operation, as shown.

```
aws-omics get-run --id run id
```

## Private workflows

Before you create and run a private workflow, you need the following resources:

- Genomics data stored in an Amazon S3 bucket or a HealthOmics sequence store.
- An Amazon S3 bucket for the workflow outputs.
- A workflow definition file (in WDL, Nextflow, or CWL).
- An Amazon ECR container image stored in a private Amazon ECR repository.

HealthOmics supports workflows written in WDL versions 1.0 and 1.1, Nextflow v22.04.0, or CWL versions 1.0, 1.1, or 1.2. To learn more about workflow languages, see the specifications for [WDL](#), [Nextflow](#), or [CWL](#).

To use a private workflow, you containerize your workflow tools and create corresponding private image repositories in Amazon Elastic Container Registry (Amazon ECR). We recommend that you define your Amazon ECR container image URIs as parameters in your workflow so that access can be verified before the run begins. It also makes it easier to run a workflow in a new Region by changing the Region parameter.

Using the HealthOmics console or API operations, you can perform the following actions related to private workflows:

- Create, retrieve, and manage workflows
- Create and manage run groups
- Share your private workflow with other AWS accounts in the same region, and accept private workflow shares offered to you by other AWS accounts

- Tag AWS resources such as workflows, runs, and run groups

For full examples of how to use HealthOmics workflows, see [HealthOmics Github tutorials](#) or [the AWS workshop end to end tutorial for HealthOmics](#).

## Topics

- [Setting up Amazon ECR for private workflows](#)
- [Writing workflow definition files](#)
- [Creating private workflows](#)
- [Sharing workflows](#)
- [Creating and working with run groups](#)

## Setting up Amazon ECR for private workflows

Before you create a private workflow, you containerize your workflow tools and create corresponding private image repositories in Amazon Elastic Container Registry (Amazon ECR). When you run the workflow, the HealthOmics service accesses the containers that you provide.

### Note

HealthOmics doesn't support ARM containers and doesn't support access to public containers.

## Topics

- [Add task inputs to an ECR container image](#)
- [Configure Amazon ECR permissions](#)

## Add task inputs to an ECR container image

Add all executables, libraries, and scripts needed to run a workflow task into the Amazon ECR image that's used to run the task.

It's best practice to avoid using scripts, binaries, and libraries that are external to a task's container image. This is especially important when using `nf-core` workflows that use a `bin` directory as part of the workflow package. While this directory will be available to the workflow task, it's mounted

as a read-only directory. Required resources in this directory should be copied into the task image and made available at runtime or when building the container image used for the task.

## Configure Amazon ECR permissions

For the HealthOmics service to access your private repository, you create an IAM policy for the HealthOmics service. You add this policy to each private repository referenced by a workflow. The private repository and workflow must be in the same region.

You can set up cross-account support to allow multiple AWS accounts (in the same region as the repository) access to the same repository.

If you share a workflow that references any Amazon ECR containers, configure cross-account support for the shared workflow subscriber to access the containers.

To configure cross-account support, give permission to specific accounts by adding a policy statement similar to `OmicsAccessCrossAccount` in the following example.

### To grant HealthOmics permission to access Amazon ECR

1. Open the [private repositories](#) page in the Amazon ECR console and select the repository you are granting access to.
2. From the side bar navigation, select **Permissions**.
3. Choose **Edit JSON**.
4. Choose **Add Statement**.
5. Add the following policy statement for **Conditions** and then select **Save Policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "omics workflow",
      "Effect": "Allow",
      "Principal": {
        "Service": "omics.amazonaws.com"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    }
  ]
}
```

```

    ]
  }
]
}

```

The resource-based policy on the registry grants HealthOmics permission to acquire a container image in the repository.

To use a cross-account container in the same region, add a permission statement similar to `OmicsAccessCrossAccount` in the following example.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "OmicsAccessPrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "omics.amazonaws.com"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    },
    {
      "Sid": "OmicsAccessCrossAccount",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{{AWS-account-ID}}:root"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    }
  ]
}

```

## Writing workflow definition files

You can write your own workflow definition file for private workflows. Workflows written in WDL, Nextflow, and CWL are supported.

### Topics

- [Writing workflows in WDL](#)
- [Writing workflows in Nextflow](#)
- [Writing workflows in CWL](#)
- [Workflow definition file examples](#)

## Writing workflows in WDL

The following tables show how inputs in WDL map to the matching primitive type or complex JSON type. Type coercion is limited and whenever possible, types should be explicit.

### Primitive types

WDL type	JSON type	Example WDL	Example JSON key and value	Notes
Boolean	boolean	Boolean b	"b": true	The value must be lower case and unquoted.
Int	integer	Int i	"i": 7	Must be unquoted.
Float	number	Float f	"f": 42.2	Must be unquoted.
String	string	String s	"s": "characters"	JSON strings that are a URI must be mapped to a WDL file to be imported.



WDL type	JSON type	Example WDL	Example JSON key and value	Notes
File	string	File f	"f": "s3:// BUCKET- NAME/path/ to/file"	Amazon S3 and HealthOmics storage URIs are imported as long as the IAM role provided for the workflow has read access to these objects. No other URI schemes are supported (such as file://, https://, and ftp://). The URI must specify an object. It cannot be a directory meaning it can not end with a /.

WDL type	JSON type	Example WDL	Example JSON key and value	Notes
Directory	string	Directory d	"d": "s3:// bucket/ path/"	The <code>Directory</code> type isn't included in WDL 1.0 or 1.1, so you will need to add <code>version development</code> to the header of the WDL file. The URI must be a Amazon S3 URI and with a prefix that ends with a '/'. All contents of the directory will be recursively copied to the workflow as a single download. The <code>Directory</code> should only contain files related to the workflow. It isn't recommended to include a <code>Directory</code> with many objects or subdirectories as it may add

WDL type	JSON type	Example WDL	Example JSON key and value	Notes
				delays while downloading.

Complex types in WDL are data structures comprised of primitive types. Data structures such as lists will be converted to arrays.

### Complex types

WDL type	JSON type	Example WDL	Example JSON key and value	Notes
Array	array	Array[Int] nums	"nums": [1, 2, 3]	The members of the array must follow the format of the WDL array type.
Pair	object	Pair[String, Int] str_to_i	"str_to_i": {"left": "a", "right": 1}	Each value of the pair must use the JSON format of its matching WDL type.
Map	object	Map[Int, String] int_to_string	"int_to_string": { 2: "hello", 1: "goodbye" }	Each entry in the map must use the JSON format of its matching WDL type.
Struct	object	struct SampleBam AndIndex { String sample_na	"b_and_i": { sample_na: "NA12878", bam:	The names of the struct members must exactly match the names of

WDL type	JSON type	Example WDL	Example JSON key and value	Notes
		<pre> me File bam File bam_index } SampleBam AndIndex b_and_i </pre>	<pre> "s3://BUC KET-NAME/ NA12878.b am", bam_index : "s3://BUC KET-NAME/ NA12878.b am.bai" } </pre>	the JSON object keys. Each value must use the JSON format of the matching WDL type.
Object	N/A	N/A	N/A	The WDL Object type is outdated and should be replaced by Struct in all cases.

The HealthOmics workflow engine does not support qualified or name-spaced input parameters. Handling of qualified parameters and their mapping to WDL parameters isn't specified by the WDL language and can be ambiguous. For these reasons, all input parameters should be declared at the top level (main) workflow and passed down to subworkflow calls by using standard WDL mechanisms.

In addition to standard `cpu`, `memory`, and `container` task runtime directives, HealthOmics also supports `acceleratorCount` and `acceleratorType`. The `acceleratorType` is dependent on the number of GPUs. It can be either a G5 instance, such as `nvidia-tesla-a10g`, or a G4 instance, such as `nvidia-tesla-t4` or `nvidia-tesla-t4-a10g`. G4 instances are not supported in the Israel (Tel Aviv) region.

A workflow definition file written in WDL that has the `acceleratorCount` and `type` defined in the parameters would look like the following.

```
version 1.1
```

```
workflow hello_gpu {
    call hello {}
    call nvidia_smi {}
}

task hello {
    command {
        echo "hello"
    }
    runtime {
    }
    output {
        String out = read_string( stdout() )
    }
}

task nvidia_smi {
    command {
        nvidia-smi
    }
    runtime {
        # Note: you will need to provision the following container image in your Amazon
        ECR Private registry
        container: "111122223333.dkr.ecr.us-west-2.amazonaws.com/nvidia/cuda:10.0-
devel-centos7"
        acceleratorCount: 1
        acceleratorType: "nvidia-tesla-t4-a10g"
    }
    output {
        String out = read_string( stdout () )
    }
}
```

## Writing workflows in Nextflow

Nextflow DSL2 is based on the Groovy programming language, so parameters are dynamic and type coercion is possible using the same rules as Groovy. Parameters and values supplied by the input JSON are available in the parameters (`params`) map of the workflow.

When an Amazon S3 or HealthOmics URI is used to construct a Nextflow file or path object, it makes the matching object available to the workflow, as long as read access is granted. The use

of prefixes or directories is allowed for Amazon S3 URIs. HealthOmics does not currently support the use of glob patterns such as “s3://BUCKET-NAME/path/\*.gz” in Amazon S3 URIs or HealthOmics Storage URIs because POSIX glob pattern behavior is undefined. Glob patterns may be used within the workflow definition in the creation of path or file channels.

HealthOmics supports Nextflow task directives `accelerator` and `type`. The `acceleratorType` is dependent on the number of GPUs. It can be either a G5 instance, such as `nvidia-tesla-a10g`, or a G4 instance, such as `nvidia-tesla-t4` or `nvidia-tesla-t4-a10g`. G4 instances aren't supported in the Israel (Tel Aviv) region.

A workflow definition file written in Nextflow that has the `accelerator` count and `type` defined in the parameters would look like the following.

```
nextflow.enable.dsl = 2

process hello {

    output:
        stdout emit: out

    script:
        """
        echo "hello"
        """

}

process nvidia_smi {
    // Note: you will need to provision the following container image in your Amazon
    ECR Private registry
    container "111122223333.dkr.ecr.<aws-region>.amazonaws.com/nvidia/cuda:10.0-devel-
centos7"
    accelerator 1, type: 'nvidia-tesla-t4-a10g'

    output:
        stdout emit: out

    script:
        """
        nvidia-smi
        """
}
```

```
}  
  
workflow HELLO_GPU {  
    hello()  
    nvidia_smi()  
}  
  
workflow {  
    HELLO_GPU()  
}
```

## Writing workflows in CWL

Workflows written in Common Workflow Language, or CWL, offer similar functionality to workflows written in WDL and Nextflow. CWL versions 1.0, 1.1, and 1.2 are supported. You can use Amazon S3 or HealthOmics storage URIs as input parameters. We recommend that you declare Amazon ECR containers in the workflow as input parameters for validation of the Amazon ECR permissions. The Amazon ECR containers must be hosted in the same Region as your workflow.

If input is defined in a `secondaryFile` in a sub workflow, the same definition must also be present in the main workflow.

To run a workflow in CWL, the following changes are required.

- All Docker container URIs should be replaced with Amazon ECR URIs.
- All the workflow files are declared in the main workflow as input, and all variables are explicitly defined.
- All JavaScript code is strict mode compliant.

CWL workflows should be defined for each container used. It isn't recommended to hardcode the `dockerPull` entry with a fixed Amazon ECR URI.

HealthOmics workflows do not support operation processes. To learn more about operation processes in CWL workflows, see the [CWL documentation](#).

HealthOmics supports GPU acceleration in CWL using `cwltool:CUDARequirement` extension syntax. The accelerator type is determined by `cudaComputeCapability`. It can be either a G5 instance, such as `nvidia-tesla-a10g`, or a G4 instance, such as `nvidia-tesla-t4`. If either G4 or G5 can be used for a

task, `nvidia-tesla-t4-a10g` is recommended. G4 instances aren't supported in the Israel (Tel Aviv) region.

It's recommended that you input your container URIs as parameters. This makes it possible to check the permissions for access to the container or for you to use the workflow in multiple Regions without having to recreate the workflow in each new Region.

The following is an example of a workflow written in CWL.

```
cwlVersion: v1.2
class: Workflow

inputs:
  in_file:
    type: File
    secondaryFiles: [.fai]

  out_filename: string
  docker_image: string

outputs:
  copied_file:
    type: File
    outputSource: copy_step/copied_file

steps:
  copy_step:
    in:
      in_file: in_file
      out_filename: out_filename
      docker_image: docker_image
    out: [copied_file]
    run: copy.cwl
```

The following file defines the `copy.cwl` task.

```
cwlVersion: v1.2
class: CommandLineTool
baseCommand: cp
```



```
inputs:
  in_file:
    type: File
    secondaryFiles: [.fai]
    inputBinding:
      position: 1

  out_filename:
    type: string
    inputBinding:
      position: 2
  docker_image:
    type: string

outputs:
  copied_file:
    type: File
    outputBinding:
      glob: "${inputs.out_filename}"

requirements:
  InlineJavascriptRequirement: {}
  DockerRequirement:
    dockerPull: "${inputs.docker_image}"
```

The following is an example of a workflow written in CWL with a GPU requirement.

```
cwlVersion: v1.2
class: CommandLineTool
baseCommand: ["/bin/bash", "docm_haplotypeCaller.sh"]
$namespaces:
  cwltool: http://commonwl.org/cwltool#
requirements:
  cwltool:CUDARequirement:
    cudaDeviceCountMin: 1
    cudaComputeCapability: "nvidia-tesla-t4"
    cudaVersionMin: "1.0"

inputs: []
outputs: []

requirements:
```

```

- class: InlineJavascriptRequirement
- class: InitialWorkDirRequirement
  listing:
  - entryname: 'docm_haplotypeCaller.sh'
    entry: |
      nvidia-smi --query-gpu=gpu_name,gpu_bus_id,vbios_version --format=csv

```

## Workflow definition file examples

The following examples are private workflow definitions for converting from CRAM to BAM in WDL. The CRAM to BAM workflow defines two tasks and uses tools from the `genomes-in-the-cloud` container, which is shown in the example and is publicly available.

Note that HealthOmics workflows require Amazon ECR containers to be in the same Region as the account calling the service. Amazon ECR containers should be included as parameters in your workflow to validate access. Workflows written in WDL don't support output.

To allow HealthOmics to access the Amazon ECR container, add the following policy to your account in the section that covers Amazon ECR repository permissions.

```

{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "omics.amazonaws.com"
      },
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ]
    }
  ]
}

```

You can include the Amazon ECR container as a parameter by including it in your workflow as shown. This is recommended so that the access permissions to your image are checked when you start the run. The following file defines all parameters for your workflow.

```
{  
  
  "input_cram": "s3://DOC-EXAMPLE-BUCKET1/inputs/NA12878.cram",  
  "ref_dict": "s3://DOC-EXAMPLE-BUCKET1/inputs/Homo_sapiens_assembly38.dict",  
  "ref_fasta": "s3://DOC-EXAMPLE-BUCKET1/inputs/Homo_sapiens_assembly38.fasta",  
  "ref_fasta_index": "s3://DOC-EXAMPLE-BUCKET1/inputs/  
Homo_sapiens_assembly38.fasta.fai",  
  "sample_name": "NA12878",  
  
  "gotc_docker": "<account_id>.dkr.ecr.<region>.amazonaws.com/genomes-in-the-  
cloud:2.4.7-1603303710"  
}
```

Then specify which files to use in your run. The following example is for when your files are stored in an Amazon S3 bucket.

```
{  
  
  "input_cram": "s3://DOC-EXAMPLE-BUCKET1/inputs/NA12878.cram",  
  "ref_dict": "s3://DOC-EXAMPLE-BUCKET1/inputs/Homo_sapiens_assembly38.dict",  
  "ref_fasta": "s3://DOC-EXAMPLE-BUCKET1/inputs/Homo_sapiens_assembly38.fasta",  
  "ref_fasta_index": "s3://DOC-EXAMPLE-BUCKET1/inputs/  
Homo_sapiens_assembly38.fasta.fai",  
  "sample_name": "NA12878"  
}
```

If you want to specify files from a sequence store, indicate that as shown in the following example, using the URI for the sequence store.

```
{  
  
  "input_cram": "omics://429915189008.storage.us-west-2.amazonaws.com/111122223333/  
readSet/4500843795/source1",  
  "ref_dict": "s3://DOC-EXAMPLE-BUCKET1/inputs/Homo_sapiens_assembly38.dict",  
  "ref_fasta": "s3://DOC-EXAMPLE-BUCKET1/inputs/Homo_sapiens_assembly38.fasta",  
  "ref_fasta_index": "s3://DOC-EXAMPLE-BUCKET1/inputs/  
Homo_sapiens_assembly38.fasta.fai",  
  "sample_name": "NA12878"  
}
```

You can then define your workflow in WDL as shown in the following.

```
version 1.0
```

```
workflow CramToBamFlow {
  input {
    File ref_fasta
    File ref_fasta_index
    File ref_dict
    File input_cram
    String sample_name
    String gotc_docker = "<account>.dkr.ecr.us-west-2.amazonaws.com/genomes-in-the-
cloud:latest"
  }
  #Converts CRAM to SAM to BAM and makes BAI.
  call CramToBamTask{
    input:
      ref_fasta = ref_fasta,
      ref_fasta_index = ref_fasta_index,
      ref_dict = ref_dict,
      input_cram = input_cram,
      sample_name = sample_name,
      docker_image = gotc_docker,
  }
  #Validates Bam.
  call ValidateSamFile{
    input:
      input_bam = CramToBamTask.outputBam,
      docker_image = gotc_docker,
  }
  #Outputs Bam, Bai, and validation report to the FireCloud data model.
  output {
    File outputBam = CramToBamTask.outputBam
    File outputBai = CramToBamTask.outputBai
    File validation_report = ValidateSamFile.report
  }
}
#Task definitions.
task CramToBamTask {
  input {
    # Command parameters
    File ref_fasta
    File ref_fasta_index
    File ref_dict
    File input_cram
    String sample_name
    # Runtime parameters
    String docker_image
```

```
}
#Calls samtools view to do the conversion.
command {
  set -eo pipefail

  samtools view -h -T ~{ref_fasta} ~{input_cram} |
  samtools view -b -o ~{sample_name}.bam -
  samtools index -b ~{sample_name}.bam
  mv ~{sample_name}.bam.bai ~{sample_name}.bai
}

#Runtime attributes:
runtime {
  docker: docker_image
}

#Outputs a BAM and BAI with the same sample name
output {
  File outputBam = "~{sample_name}.bam"
  File outputBai = "~{sample_name}.bai"
}
}

#Validates BAM output to ensure it wasn't corrupted during the file conversion.
task ValidateSamFile {
  input {
    File input_bam
    Int machine_mem_size = 4
    String docker_image
  }
  String output_name = basename(input_bam, ".bam") + ".validation_report"
  Int command_mem_size = machine_mem_size - 1
  command {
    java -Xmx~{command_mem_size}G -jar /usr/gitc/picard.jar \
    ValidateSamFile \
    INPUT=~{input_bam} \
    OUTPUT=~{output_name} \
    MODE=SUMMARY \
    IS_BISULFITE_SEQUENCED=false
  }
  runtime {
    docker: docker_image
  }
  #A text file is generated that lists errors or warnings that apply.
```

```
output {
  File report = "~{output_name}"
}
}
```

For workflows written in Nextflow, you must define a **publishDir** directive to export task content to your output Amazon S3 bucket. As shown in the following example, set the **publishDir** value to `/mnt/workflow/pubdir`. To export files to Amazon S3, the files must be in this directory.

```
nextflow.enable.dsl=2

workflow {
  CramToBamTask(params.ref_fasta, params.ref_fasta_index, params.ref_dict,
  params.input_cram, params.sample_name)
  ValidateSamFile(CramToBamTask.out.outputBam)
}

process CramToBamTask {
  container "<account>.dkr.ecr.us-west-2.amazonaws.com/genomes-in-the-cloud"

  publishDir "/mnt/workflow/pubdir"

  input:
    path ref_fasta
    path ref_fasta_index
    path ref_dict
    path input_cram
    val sample_name

  output:
    path "${sample_name}.bam", emit: outputBam
    path "${sample_name}.bai", emit: outputBai

  script:
  """
    set -eo pipefail

    samtools view -h -T $ref_fasta $input_cram |
    samtools view -b -o ${sample_name}.bam -
    samtools index -b ${sample_name}.bam
    mv ${sample_name}.bam.bai ${sample_name}.bai
  """
}
```

```
process ValidateSamFile {
  container "<account>.dkr.ecr.us-west-2.amazonaws.com/genomes-in-the-cloud"

  publishDir "/mnt/workflow/pubdir"

  input:
    file input_bam

  output:
    path "validation_report"

  script:
    """
    java -Xmx3G -jar /usr/gitc/picard.jar \
    ValidateSamFile \
    INPUT=${input_bam} \
    OUTPUT=validation_report \
    MODE=SUMMARY \
    IS_BISULFITE_SEQUENCED=false
    """
}
```

## Creating private workflows

When you create a private workflow, you specify the following resources:

- Input data location: an Amazon S3 location or a HealthOmics storage URI.
- An Amazon S3 location for the workflow outputs.
- An Amazon ECR container image stored in a private Amazon ECR repository.
- An IAM policy that grants the workflow access to the preceding resources.

### Topics

- [IAM policy to give workflow access to resources](#)
- [Workflow definition files](#)
- [Parameter templates](#)
- [Create a private workflow](#)
- [Verify the status of your workflow](#)

- [Workflow tasks](#)

## IAM policy to give workflow access to resources

The following is a comprehensive example of an IAM role that grants permission to access those resources. This policy also includes access to some Amazon CloudWatch logs that can help with troubleshooting or tracking the use of AWS actions and resources. The CloudWatch permissions aren't required to run a workflow.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET2/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```



```

        "logs:DescribeLogStreams",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:us-west-2:123456789012:log-group:/aws/omics/
WorkflowLog:log-stream:*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup"
    ],
    "Resource": [
        "arn:aws:logs:us-west-2:{{accountId}}:log-group:/aws/omics/
WorkflowLog:*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchCheckLayerAvailability"
    ],
    "Resource": [
        "arn:aws:ecr:us-west-2:{{accountId}}:repository/*"
    ]
}
]
}

```

Authorize the service to use the role, by adding the following trust policy.

```

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

```

```
    }  
  }  
]  
}
```

## Workflow definition files

The HealthOmics workflow definition files must meet the following requirements:

- HealthOmics supports workflow definitions written in WDL, Nextflow, or CWL.
- Declare all parameters in the workflow definition file. Parameters include input and output locations, Amazon ECR container repositories, and runtime parameters such as allocated memory or CPU.

### Note

The storage requirements to perform runs may be more than expected due to internal file system usage, so allow for more allocated memory than anticipated in your workflow definition file.

- Declare the output files in the workflow definition file. If you want to copy intermediate run files to the output location, declare them as workflow outputs.

The input and output locations must be in the same Region as the workflow run.

- HealthOmics storage workflow inputs must be in ACTIVE status. OM will not import inputs with an ARCHIVED status, causing the workflow to fail.

The following is an example WDL workflow that reads the contents of an INPUT file and writes them into a RESULT file.

```
version 1.0  
  
workflow TestFlow {  
  input {  
    File input_txt_file  
  }  
  
  # Copies input file data to output.  
  call TxtFileCopyTask{
```

```
    input:
      input_txt_file = input_txt_file,
  }

  output {
    File output_txt_file = TxtFileCopyTask.output_txt_file
  }
}

# Task definitions.
task TxtFileCopyTask {
  input {
    File input_txt_file
  }

  command {
    cat ~{input_txt_file} > outfile.txt
  }

  output {
    File output_txt_file = "outfile.txt"
  }

  runtime {
    cpu: 2
    memory: "4 GiB"
    docker: "ACCOUNT-ID.dkr.ecr.us-west-2.amazonaws.com/ubuntu:latest"
  }
}
```

The `input_txt_file.json` file contains the following content:

```
{
  "input_txt_file": {
    "description": "Input file to be copied",
    "required": true
  }
}
```

You must zip the workflow definition file and any dependencies, such as subworkflows, before you can use the file to create a workflow with the **create-workflow** API operation.

## Parameter templates

When creating a workflow, create a parameter template JSON file if the workflow has required inputs. Each input is a named object where the name must match the exact name of the workflow input. Each object must have a description string which the service console displays in the **Start run** page. Each object may declare a boolean `optional` value indicating if the value is optional for all runs. If a parameter isn't marked as `optional`, the default value is `true`, as shown in the following example parameter template in JSON.

```
{
  "myRequiredParameter1": {
    "description": "this parameter is required",
  },
  "myRequiredParameter2": {
    "description": "this parameter is also required",
    "optional": false
  },
  "myOptionalParameter": {
    "description": "this parameter is optional",
    "optional": true
  }
}
```

A workflow written in CWL does not require a parameter template. HealthOmics auto-detects all the workflow inputs for CWL workflows.

After you define your workflow and the parameters, you can create a workflow using the CLI as shown. If you are including multiple workflow definition files, use the `--main` parameter to specify which file is the main definition file for your workflow. You can also specify an accelerator.

```
aws omics create-workflow
  --name Test --main multi_workflow/workflow2.wdl
  --definition-zip fileb://definition.zip
  --parameter-template file://params_sample_description.json
  --accelerators GPU
```

You receive the following response when the workflow is successfully created.

```
{
  "arn": "arn:aws:omics:us-west-2:....",
```

```
"id": "1234567",
"status": "CREATING",
"tags": {
  "resourceArn": "arn:aws:omics:us-west-2:...."
}
}
```

Larger zip files containing the workflow definition can be loaded from an Amazon S3 bucket using the `--definition-uri` parameter.

## Create a private workflow

When you create a workflow, you specify workflow definitions and parameters that the engine uses to run the workflow.

### Create a private workflow (console)

#### To create a private workflow

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Private workflows**.
3. On the **Private workflows** page, choose **Create workflow**.
4. On the **Create workflow** page, provide the following information
  - **Workflow name** - A distinctive name for this workflow.
  - **Description** (optional) - A description of this workflow.
  - **Default run storage capacity** (optional) - The default amount of run storage required for this workflow. The default value is 1.2 TB. You can override this default when you start a workflow run.
  - Under **Workflow definition**, choose **Select definition folder from S3**.
  - For **Workflow definition in S3**, enter the Amazon S3 location that contains the workflow definition.
  - For **Workflow language**, select the specification language of the workflow.
  - **Tags** (optional) - Provide up to 50 tags for this workflow.
5. Choose **Next**.
6. On the **Add workflow parameters** page, provide the workflow parameters. You can either upload a JSON file that specifies the parameters or manually enter your workflow parameters.

7. Choose **Next**.
8. Review the workflow configuration, then choose **Create workflow**.

## Create a private workflow (API)

You can create a workflow with the accelerators parameter defined, as shown.

```
aws omics create-workflow --name workflow name \  
  --definition-uri s3://DOC-EXAMPLE-BUCKET1/GPUWorkflow.zip \  
  --accelerators GPU
```

An HealthOmics reference store object can be referred to with a URI like the following. Use your own *account ID*, *reference store ID*, and *reference ID* where indicated.

```
omics://account ID.storage.us-west-2.amazonaws.com/reference store id/reference/id
```

Some workflows will require both the SOURCE and INDEX files for the reference genome. The previous URI is the default short form and will default to the SOURCE file. In order to specify either file, you can use the long URI form, as follows.

```
omics://account ID.storage.us-west-2.amazonaws.com/reference store id/reference/id/  
source  
  omics://account ID.storage.us-west-2.amazonaws.com/reference store id/reference/id/  
index
```

Using a sequence read set would have a similar pattern, as shown.

```
aws omics create-workflow \  
  --name workflow name \  
  --main sample workflow.wdl \  
  --definition-uri omics://account ID.storage.us-  
west-2.amazonaws.com/sequence_store_id/readSet/id \  
  --parameter-template file://parameters_sample_description.json
```

Some read sets, such as those based on FASTQ, can contain paired reads. In the following examples, they're referred to as SOURCE1 and SOURCE2. Formats such as BAM and CRAM will only have a SOURCE1 file. Some read sets will contain INDEX files such as bai or crai files. The

preceding URI is the default short form and will default to the SOURCE1 file. To specify the exact file or index, you can use the long URI form, as follows.

```
omics://123456789012.storage.us-west-2.amazonaws.com/<sequence_store_id>/readSet/<id>/
source1
  omics://123456789012.storage.us-west-2.amazonaws.com/<sequence_store_id>/readSet/
<id>/source2
  omics://123456789012.storage.us-west-2.amazonaws.com/<sequence_store_id>/readSet/
<id>/index
```

The following is an example of an input JSON file that uses two Omics Storage URIs.

```
{
  "input_fasta": "omics://123456789012.storage.us-west-2.amazonaws.com/
<reference_store_id>/reference/<id>",
  "input_cram": "omics://123456789012.storage.us-west-2.amazonaws.com/
<sequence_store_id>/readSet/<id>"
}
```

Reference the input JSON file in the AWS CLI by adding `--inputs file://<input_file.json>` to your `start-run` request.

## Verify the status of your workflow

After you create your workflow, you can verify the status and view other details of the workflow using `get-workflow`, as shown.

```
aws omics get-workflow --id 1234567
```

The response gives you your workflow details, including the status, as shown.

```
{
  "arn": "arn:aws:omics:us-west-2:....",
  "id": "1234567",
  "status": "ACTIVE",
  "type": "PRIVATE",
  "name": "workflow_name"
  "creationTime": "2022-07-06T00:27:05.542459"
}
```

Before a run can be started, the status must be listed as ACTIVE.

## Workflow tasks

Workflow tasks are the individual processes within a run. Each task has a unique identifier. For a private workflow, HealthOmics workflows will use your defined compute specifications to run your task. Input files to the workflow and workflow tasks are staged to a scratch volume that's dedicated to the workflow run. They are read-only, which prevents tasks modifying potential inputs to other tasks in a workflow. The directories included as input are also read-only. All inputs are made available to the tasks' working directory as symbolic links. They're only accessible if they're declared in the workflow definition file. Many genomics applications assume that index files will be in the same location as a sequence file (such as a companion `bai` file for a `bam` file). To make sure indexes are present, you must specify them as tasks inputs.

Because workflow tasks can't connect to the public internet, they can't download resources by using `http`, `https`, or `ftp`. Required resources should be included as workflow inputs from Amazon S3 or an HealthOmics sequence store. They also should be present in the container images that are used to run workflow tasks. Workflow tasks may interact with Amazon S3 as long as the IAM role used to run the workflow has been granted access.

### Memory and computation considerations for tasks

Private workflow tasks are run on HealthOmics instances by using the smallest instance that can accommodate the requested CPU and memory. We recommend choosing the most sensible combination of CPU and memory for your needs. For example, if you need 64 GiB of RAM, then the most cost-effective type is `omics.r.2xlarge`. This type allocates eight vCPUs. If your task allocation only calls for one vCPU, the task container only gets one, even though eight are available on the host machine. Unless this will adversely affect the workflow, you might want to set a larger number of vCPUs. All tasks reserve a small amount of memory for management and logging agents, so the full memory allocation might not always be available to the application in the task.

Container resource allocations are hard limits. Tasks that run out of memory or attempt to use additional vCPUs can be immediately shut down by the host, potentially without warning.

### Running Java applications in a private workflow task

When running Java applications in a workflow task, the image used to run the task must contain Java 1.8–212 or later. Versions before this might attempt to allocate heap memory that's not available to the container. By default, for versions after 1.8-212, the heap allocation of the JVM will be 25% of the memory available to the container. If you use a `-Xmx` flag to request a larger amount, be aware that not all memory consumed by Java is heap memory. Allocating 100% of the available memory to the JVM heap causes the task to fail.



## Including task inputs in Amazon ECR images

All executables, libraries, and scripts needed to run a workflow task should be provided by the Amazon ECR image that's used to run the task.

It's best practice to avoid using scripts, binaries, and libraries that are external to a task's container image. This is especially important when using `nf-core` workflows that use a `bin` directory as part of the workflow package. While this directory will be available to the workflow task, it's mounted as a read-only directory. Required resources in this directory should be copied into the task image and made available at runtime or when building the container image used for the task.

## Debugging workflow tasks

The following are best practices and considerations for debugging your tasks and workflows.


- Task logs rely on `STDOUT` and `STDERR` being produced by the task. If the application used in the task doesn't produce either of these, then there won't be a task log. To assist with debugging, use applications in `verbose` mode.
- To view the commands being run in a task along with their interpolated values, use the `set -x` Bash command. This can help determine if the task is using the correct inputs and identify where errors might have kept the task from running as intended.
- Use the `echo` command to output the values of variables to `STDOUT` or `STDERR`. This helps you confirm that they're being set as expected.
- Use commands like `ls -l <name_of_input_file>` to confirm that inputs are present and are of the expected size. If they aren't, this might reveal a problem with a prior task producing empty outputs due to a bug.
- Use the command `df -Ph . | awk 'NR==2 {print $4}'` in a task's script to determine the space currently available to the task and help identify situations where you might need to run the workflow with additional storage allocation.

Including any of the preceding commands in a task script assumes that the task container also includes these commands and that they are on the path of the container environment.

## Sharing workflows

As the owner of a private workflow, you can share the workflow with an AWS account in the same region. To share a workflow with more than one AWS account, you create multiple shares of the same workflow.

As the owner, you can revoke access to a shared workflow by deleting the share.

 **Note**

To share the Amazon ECR containers associated with a shared workflow, you need to adjust the container permissions to allow cross-account access. For more information, including an example policy, see [Amazon ECR permissions](#).

To subscribe to a shared workflow, you follow these steps to accept and use the workflow:

1. Use the console or API to accept the share. Set your current region to the same region as the share request.
  - To find the share request in the console, navigate to the **All Resource shares** page, then choose the **Shared with me** tab.
2. Use the console or API to create a run for the shared workflow.
  - To find the workflow details page in the console, navigate to **Shared with me** (see step 1), then choose the **Resource link** for the shared workflow.
3. You provide your own input data for the workflow.
4. The shared workflow runs in your AWS account.

As the subscriber to a shared workflow, the system blocks you from performing the following workflow actions:

- Exporting a shared workflow
- Re-running the shared workflow
  - You create a new run for the shared workflow.
- Re-sharing the workflow.
- Assigning a tag to the workflow.
- Deleting the workflow.
  - When you no longer need the workflow, you delete the workflow share.

See [Cross-account resource sharing in AWS HealthOmics](#) for additional information about resource sharing.

## Topics

- [Share a private workflow \(console\)](#)
- [Share a private workflow \(API\)](#)
- [Accept a shared workflow \(console\)](#)
- [Run a shared workflow \(console\)](#)
- [Run a shared workflow \(API\)](#)

## Share a private workflow (console)

From the console, you can share a private workflow with an AWS account in the same region as the workflow.

### To share a private workflow

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Private workflows**.
3. From the **Workflows** table on the **Private workflows** page, select the workflow to share, and choose **Share**.
4. In the **Share details** panel of the **Share workflow** page, enter a descriptive name for the share and enter the AWS account of the subscriber.
5. Choose **Share resource**. The console displays resource shares in the **All resource shares** page.

The initial state of the share is pending. After the subscriber accepts the share, the state changes to active.

## Share a private workflow (API)

Use the **create-share** API operation to create a workflow share. The principal subscriber is the AWS account of the user who will get access to the workflow.

```
aws omics create-share \  
  --resource-arn "arn:aws:omics:us-west-2:555555555555:workflow/123456" \  
  --principal-subscriber "123456789012" \  
  --name "my_Share-123"
```

If the create is successful, you receive a response with the share ID and status.

```
{
  "shareId": "495c21bedc889d07d0ab69d710a6841e-dd75ab7a1a9c384fa848b5bd8e5a7e0a",
  "name": "my_Share-123",
  "status": "PENDING"
}
```

The share remains in pending state until the subscriber accepts it using the `accept-share` API operation.

See [Cross-account resource sharing in AWS HealthOmics](#) for other API usage examples.

## Accept a shared workflow (console)

You can use the console to accept an offered workflow share. Make sure to set the console to the same Region as the workflow.

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **All Resource shares**, then choose the **Shared with me** tab.
3. From the **Resources shared with me** table, select the workflow share and then choose **Accept**.

After you accept the workflow, choose the **Resource link** for the shared workflow to view its details.

## Run a shared workflow (console)

After you accept a workflow share, you can start a run on the workflow.

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **All Resource shares**, then choose the **Shared with me** tab.
3. From the **Resources shared with me** table, choose the **Resource link** for the shared workflow.
4. In the **Workflow details** page, choose **Create run**.

The console opens the **Create run** page, with the workflow type (shared) and **Workflow ID** pre-populated.

5. Configure the remaining fields in the **Create run** form. For additional information, see [Starting runs \(console\)](#).

## Run a shared workflow (API)

Use `get-workflow` to retrieve the ARN of the shared workflow.

```
aws omics get-workflow --id 1234567 \  
--workflow-owner-id 5555555555
```

When you run the workflow, provide the workflow owner's AWS account ID and the ARN of the shared workflow.

```
aws omics start-run --id 1234567 --workflow-owner-id 5555555555 \  
--role-arn arn:aws:iam::1234567892012:role/service-role/OmicsWorkflow-20221004T164236 \  
--name ArchiveTest --retention-mode REMOVE
```

## Creating and working with run groups

You can optionally create a run group to limit the compute resources for the runs that you add to the group. Run groups can help you:

- Queue your runs so that you don't exceed service limits.
- Catch run-away tasks by setting a maximum run duration limit.
- Manage the priority of each run so that the most important runs complete first

If you set the maximum concurrent vCPU, GPU, or runs, run tasks will queue when the limit is reached. If you set a maximum run duration, the run fails if it exceeds the maximum duration.

Use the run priority setting to establish priority within a run group.

Service limits take precedence over run group limits. For instance, if a run group limit is set higher than your service limit, the service limit will apply first.

### Run priority

You can use run priority to establish the priority of runs in a run group.

If multiple runs have the same priority, the run that started first has the higher priority.

You can also set a priority for a run that isn't in a run group. The priority is compared with the priorities of all other runs that aren't in a run group

You set run priority when you start the workflow run. For more information, see [Starting a workflow run](#).

## Create a run group (console)

### To create a run group

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Run groups**.
3. On the **Run groups** page, choose **Create run group**.
4. On the **Create run group details** page, provide the following information
  - **Run group name** - A unique name for this run group.
  - **Max vCPU for concurrent runs** - The maximum number of vCPUs that can run concurrently across all active runs in the run group.
  - **Max GPUs** - The maximum number of GPUs that can run concurrently across all active runs in the run group.
  - **Max run time (mins) per run** - The maximum time for each run (in minutes). If a run exceeds the maximum run time, the run fails automatically.
  - **Max concurrent runs** - The maximum number of runs that can be running at the same time.
5. (optional) You can add up to 50 **tags** to the run group.
6. Choose **Create run group**.

## Create a run group (API)

To create a run group, use the **create-run-group** API operation to create a run group named TestRunGroup. The following example sets a maximum of 20 CPUs, 10 GPUs, 5 runs, and a maximum run duration of 600 minutes.

```
aws omics create-run-group --name TestRunGroup \  
--max-cpus 20 \  
--max-gpus 10 \  
--max-runs 5 --max-run-time 600
```

```
--max-duration 600 \  
--max-runs 5
```

The response from this API operation includes the ID of the newly created RunGroup.

```
{  
  "arn": "arn:aws:omics:us-west-2:12345678901:runGroup/2839621",  
  "id": "2839621",  
  "tags": {}  
}
```

To get additional information about the run group, use this ID with the **get-run-group** API operation, as shown in the following example.

```
aws omics get-run-group --id run group id
```

The response includes the limit settings for the run group and the assigned tags.

```
{  
  "arn": "arn:aws:omics:us-west-2:776893852117:runGroup/2839621",  
  "id": "2839621",  
  "name": "TestRunGroup",  
  "maxCpus": 20,  
  "maxRuns": 5,  
  "maxDuration": 600,  
  "creationTime": "2024-06-12T15:35:39.191730+00:00",  
  "tags": {},  
  "maxGpus": 10  
}
```

You can also use the **list-run-group** API operation to view all created run groups.

```
aws omics list-run-groups
```

## Running workflows

After you create your workflow, you can perform runs either individually or as part of a run group. The following topics show how to create an optional run group to limit computational resources

and reduce costs. You can also use a run group to queue runs to be processed. There are also examples of how to start runs and get information on ongoing runs.

When you start a run, HealthOmics allocates temporary run storage, because workflow engines expect to have access to scratch storage during the run. To ensure data isolation and security, HealthOmics provisions and deprovisions the storage for each run.

## Topics

- [Run storage types](#)
- [Starting a workflow run](#)
- [Deleting workflows and runs](#)
- [Define custom IAM permissions for runs](#)

## Run storage types

For a given workflow or workflow run, you can choose static or dynamic run storage. By default, HealthOmics provides static run storage. Consider the following factors when deciding which run storage type to use:

- Static
  - HealthOmics allocates a fixed amount of run storage.
  - You can specify the storage size in the StartRun API request. The system rounds up the value to the nearest multiple of 1200 GiB. If that storage size isn't available, it rounds up to the nearest multiple of 2400 GiB.
  - The default run storage is 1200 GiB, if you don't specify a value.
  - If the specified storage size is too low, the run fails with an **Out of storage for file system** error.
  - Static run storage is suitable for large workflows. It provides higher file system throughput per GiB and lower cost per GiB than dynamic run storage.
  - Use static run storage for burst workloads that scale out wide and quickly (for example, a large volume of RNASeq samples processed in parallel).
- Dynamic
  - You don't need to estimate the required storage for the run. HealthOmics allocates a starting amount of run storage. The storage size dynamically scales up and down, based on file system utilization during the run. A run never fails due to an **Out of storage for file system** error.



- Dynamic run storage provides faster provisioning/deprovisioning time than static run storage. Faster setup is an advantage for smaller workflows that run frequently and is also an advantage during development/test cycles.
- Dynamic run storage uses burst credits to control burst throughput, so don't use it for workflows that require a peak burst throughput of 50MiBs or higher.
- When burst credits expire, dynamic run storage capacity increases can slow down. The system creates a warning in the logs when a burst credit expires. If your workflow frequently runs out of burst credits, consider using static run storage.
- After the run completes (success path or fail path), the getRun API operation returns the maximum storage used by the run in the storageCapacity field. You can also find this information in the run manifest logs located in the **omics** log group.
  - For a dynamic storage run that completes within 2 hours, the maximum storage value may not be available.

#### Note

Run storage usage incurs charges to your account. For pricing information about static and dynamic run storage, see [HealthOmics pricing](#).

## Calculating required static run storage

A workflow requires additional capacity when it uses static run storage (compared with dynamic run storage) because the base file system installation uses 7% of the static file system capacity.

If you run a dynamic run storage workflow to measure the maximum storage used by the run, use the following calculation to determine the minimum amount of static storage required:

```
static storage required =  
    maximum storage in GiB used by the dynamic run storage  
    + (total static file system size in GiB * 0.07)
```

For example:

```
Maximum storage measured from a dynamic run storage workflow run: 500GiB  
File system size: 1200GiB  
7% of the file system size: 84GiB
```

```
500 + 84 = 584GiB of static run storage required for this run.
```

Therefore, 1200GiB (the minimum capacity for static run storage) is sufficient for this run.

## Starting a workflow run

When you start a run, you can set the run storage type and storage amount (for static storage). For additional information, see [Run storage types](#).

You also set the run priority. How priority impacts the run depends on whether the run is associated with a run group. For additional information, see [Run priority](#).

## Starting runs (console)

### To start a workflow run

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. In the left navigation pane, choose **Runs**.
3. On the **Runs** page, choose **Create run**.
4. On the **Create run** page, provide the following information
  - **Workflow ID** - The workflow ID associated with this run.
  - **Run name** - A distinctive name for this run.
  - **Run priority** - The priority of this run. Higher numbers specify a higher priority, and the highest priority tasks are run first.
  - **Run storage capacity** - The amount of temporary storage needed for the run. By default, the run storage capacity that was set for the workflow will be selected. You can select a different run storage capacity for your run.
  - **Select S3 output destination** - The S3 location where the run outputs will be saved.
5. Under **Service role**, you can use an existing service role or create a new one.
6. (Optional) For **Tags**, you can assign up to 50 tags to the run.
7. Choose **Next**.
8. On the **Add parameter values** page, provide the workflow parameters. You can either upload a JSON file that specifies the parameters or manually enter your workflow parameters.
9. Choose **Next**.
10. On the **Add run groups** page, provide the run group details.

11. On the **Run cache** page, provide the run cache details.
12. Choose **Review and start run**.
13. On the **Review and start run** page, choose **Start run**.

## Starting runs (API)

Use the **start-run** API operation with the IAM role and Amazon S3 bucket that you created. Although the default retention mode is RETAIN, this example sets the retention mode to REMOVE. If the quota for maximum runs has been met, the earliest runs with REMOVE retention mode are deleted first. This makes room for new runs to start—even if the maximum runs limit is met—as long as there are runs with REMOVE retention mode that can be removed.

When the parameter is set to REMOVE, the run metadata is removed after the run completes and the metadata has been sent to Amazon CloudWatch.

```
aws omics start-run
  --workflow-id workflow id \
  --role-arn arn:aws:iam::1234567892012:role/service-role/
OmicsWorkflow-20221004T164236 \
  --name workflow name \
  --retention-mode REMOVE
```

In response, you get the following output. The `uuid` is unique to the run, and along with `runOutputUri` can be used to track where output data is written.

```
{
  "arn": "arn:aws:omics:us-west-2:....:run/1234567",
  "id": "1234567",
  "uuid": "96c57683-74bf-9d6d-ae7e-f09b097db14a",
  "runOutputUri": "s3://bucket/folder/8405154/96c57683-74bf-9d6d-ae7e-f09b097db14a"
  "status": "PENDING"
}
```

If the parameter template for a workflow declares any required parameters, you can provide a local JSON file of the inputs when you start a workflow run. The JSON file contains the exact name of each input parameter and a value for the parameter.

Reference the input JSON file in the AWS CLI by adding `--inputs file://<input_file.json>` to your `start-run` request.

You can also use the **start-run** API with a GPU workflow ID, as shown.

```
aws omics start-run --workflow-id workflow id \  
  --role-arn arn:aws:iam::1234567892012:role/service-role/  
OmicsWorkflow-20221004T164236 \  
  --name GPUPTestRunModel \  
  --output-uri s3://DOC-EXAMPLE-BUCKET1
```

## Get information about a workflow run

You can use the ID in the response with the **get-run** API to check the status of a run, as shown.

```
aws omics get-run --id run id
```

The response from this API operation tells you the status of the workflow run. Possible statuses are PENDING, STARTING, RUNNING, and COMPLETED. When a run is COMPLETED, you can find an output file called `outfile.txt` in your output Amazon S3 bucket, in a folder named after the run ID.

The **get-run** API operation also returns other details, such as whether the workflow is Ready2Run or PRIVATE, the workflow engine, and accelerator details. The following example shows the response for **get-run** for a run of a private workflow, described in WDL with a GPU accelerator and no tags assigned to the run.

```
{  
  "arn": "arn:aws:omics:us-west-2:123456789012:run/7830534",  
  "id": "7830534",  
  "uuid": "96c57683-74bf-9d6d-ae7e-f09b097db14a",  
  "runOutputUri": "s3://bucket/folder/8405154/96c57683-74bf-9d6d-ae7e-f09b097db14a"  
  "status": "COMPLETED",  
  "workflowId": "4074992",  
  "workflowType": "PRIVATE",  
  "roleArn": "arn:aws:iam::123456789012:role/service-role/  
OmicsWorkflow-20221004T164236",  
  "name": "RunGroupMaxGpuTest",  
  "runGroupId": "9938959",  
  "digest":  
  "sha256:a23a6fc54040d36784206234c02147302ab8658bed89860a86976048f6cad5ac",  
  "accelerators": "GPU",  
  "outputUri": "s3://DOC-EXAMPLE-BUCKET1",  
  "startedBy": "arn:aws:sts::123456789012:assumed-role/Admin/<role_name>",
```

```
"creationTime": "2023-04-07T16:44:22.262471+00:00",
"startTime": "2023-04-07T16:56:12.504000+00:00",
"stopTime": "2023-04-07T17:22:29.908813+00:00",
"tags": {}
}
```

You can see the status of all runs with the **list-runs** API operation, as shown.

```
aws omics list-runs
```

To see all the tasks completed for a specific run, use the **list-run-tasks** API.

```
aws omics list-run-tasks --id task ID
```

To get the details of any specific task, use the **get-run-task** API.

```
aws omics get-run-task --id <run_id> --task-id task ID
```

After the run completes, the metadata is sent to CloudWatch under the stream **manifest/run/<run ID>/<run UUID>**.

The following is an example of the manifest.

```
{
  "arn": "arn:aws:omics:us-east-1:123456789012:run/1695324",
  "creationTime": "2022-08-24T19:53:55.284Z",
  "resourceDigests": {
    "s3://omics-data/broad-references/hg38/v0/Homo_sapiens_assembly38.dict":
      "etag:3884c62eb0e53fa92459ed9bfff133ae6",
    "s3://omics-data/broad-references/hg38/v0/Homo_sapiens_assembly38.fasta":
      "etag:e307d81c605fb91b7720a08f00276842-388",
    "s3://omics-data/broad-references/hg38/v0/Homo_sapiens_assembly38.fasta.fai":
      "etag:f76371b113734a56cde236bc0372de0a",
    "s3://omics-data/intervals/hg38-mjs-whole-chr.500M.intervals":
      "etag:27fdd1341246896721ec49a46a575334",
    "s3://omics-data/workflow-input-lists/dragen-gvcf-list.txt":
      "etag:e22f5aeed0b350a66696d8ffae453227"
  },
  "digest":
    "sha256:a5baaff84dd54085eb03f78766b0a367e93439486bc3f67de42bb38b93304964",
  "engine": "WDL",
  "main": "gatk4-basic-joint-genotyping-v2.wdl",
}
```

```

    "name": "1044-gvcfs",
    "outputUri": "s3://omics-data/workflow-output",
    "parameters": {
      "callset_name": "cohort",
      "input_gvcf_uris": "s3://omics-data/workflow-input-lists/dragen-gvcf-list.txt",
      "interval_list": "s3://omics-data/intervals/hg38-mjs-whole-chr.500M.intervals",
      "ref_dict": "s3://omics-data/broad-references/hg38/v0/
Homo_sapiens_assembly38.dict",
      "ref_fasta": "s3://omics-data/broad-references/hg38/v0/
Homo_sapiens_assembly38.fasta",
      "ref_fasta_index": "s3://omics-data/broad-references/hg38/v0/
Homo_sapiens_assembly38.fasta.fai"
    },
    "roleArn": "arn:aws:iam::123456789012:role/OmicsServiceRole",
    "startedBy": "arn:aws:sts::123456789012:assumed-role/admin/ahenroid-Isengard",
    "startTime": "2022-08-24T20:08:22.582Z",
    "status": "COMPLETED",
    "stopTime": "2022-08-24T20:08:22.582Z",
    "storageCapacity": 9600,
    "uuid": "a3b0ca7e-9597-4ecc-94a4-6ed45481aeab",
    "workflow": "arn:aws:omics:us-east-1:123456789012:workflow/1558364",
    "workflowType": "PRIVATE"
  },
  {
    "arn": "arn:aws:omics:us-east-1:123456789012:task/1245938",
    "cpus": 16,
    "creationTime": "2022-08-24T20:06:32.971290",
    "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/gatk",
    "imageDigest":
"sha256:8051adab0ff725e7e9c2af5997680346f3c3799b2df3785dd51d4abdd3da747b",
    "memory": 32,
    "name": "geno-123",
    "run": "arn:aws:omics:us-east-1:123456789012:run/1695324",
    "startTime": "2022-08-24T20:08:22.278Z",
    "status": "SUCCESS",
    "stopTime": "2022-08-24T20:08:22.278Z",
    "uuid": "44c1a30a-4eee-426d-88ea-1af403858f76"
  },
  ...

```

Run metadata isn't deleted if it's not present in the CloudWatch logs. You can also use the run ID to rerun workflow runs using the CLI tool. Learn more and download the tool from the [HealthOmics Tool Github repository](#).

## Re-running a workflow run

The following is an example of using the tool to rerun a workflow run, using the run ID. You can retrieve an ID for a run the CloudWatch logs.

```
omics-rerun 9876543 --name workflow name --retention-mode REMOVE
```

If the run exists in CloudWatch, you receive a response similar to the following.

Original request:

```
{
  "workflowId": "9679729",
  "roleArn": "arn:aws:iam::123456789012:role/DemoRole",
  "name": "sample_rerun",
  "parameters": {
    "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/default:latest",
    "file1": "omics://123456789012.storage.us-west-2.amazonaws.com/8647780323/readSet/6389608538"
  },
  "outputUri": "s3://workflow-output-bcf2fcb1"
}
```

StartRun request:

```
{
  "workflowId": "9679729",
  "roleArn": "arn:aws:iam::123456789012:role/DemoRole",
  "name": "new test",
  "parameters": {
    "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/default:latest",
    "file1": "omics://123456789012.storage.us-west-2.amazonaws.com/8647780323/readSet/6389608538"
  },
  "outputUri": "s3://workflow-output-bcf2fcb1"
}
```

StartRun response:

```
{
  "arn": "arn:aws:omics:us-west-2:123456789012:run/9171779",
  "id": "9171779",
  "status": "PENDING",
  "tags": {}
}
```

If the workflow no longer exists, you receive an error message.

## Deleting workflows and runs

When you no longer need a workflow, run, or run group, you can delete it by using the AWS CLI, API, or console. A workflow can only be deleted when it's listed in ACTIVE or FAILED status and has no active shares. Deleting a workflow does not affect any ongoing runs that are using the workflow.

The following example shows how you can use the AWS CLI command to delete a workflow. You won't receive a response. To run the example, replace the *workflow id* with the ID of the workflow you want to delete.

```
aws omics delete-workflow --id workflow id
```

In addition to deleting a run, you can also cancel a run. To cancel a run, its status must be PENDING, STARTING, RUNNING, or STOPPING. The following AWS CLI command shows how you can cancel a run. To run the example, replace the *run id* with the ID of the run you would like to cancel. If successful, there is no response.

```
aws omics cancel-run --id run id
```

The following AWS CLI command deletes a run. Runs can only be deleted if they are complete or canceled. To run the example, replace the *run id* with the ID of the run you want to delete. There is no response if the run is successfully deleted.

```
aws omics delete-run --id run id
```

You can also delete run groups. Run groups can only be deleted if there are no runs associated with that run group with the status of PENDING, STARTING, RUNNING, or STOPPING.

The following example shows how you can use the AWS CLI to delete a run group. You will not receive a response. To run the example, replace the *run group id* with the ID of the run group you want to delete.

```
aws omics delete-run-group --id run group id
```

## Define custom IAM permissions for runs

You can include any workflow, run, or run group referenced by the StartRun request in an authorization request. To do so, list the desired combination of workflows, runs, or run groups in



the IAM policy. For example, you can limit the use of a workflow to a specific run or run group. You can also specify that a workflow only be used with a run group.

The following is an example IAM policy that allows a single workflow with a single run group.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "omics:StartRun"
      ],
      "Resource": [
        "arn:aws:omics:us-west-2:123456789012:workflow/1234567",
        "arn:aws:omics:us-west-2:123456789012:runGroup/2345678"
      ]
    },
    {
      # Optionally, allow user to rerun a failed run.
      "Effect": "Allow",
      "Action": [
        "omics:StartRun"
      ],
      "Resource": [
        "arn:aws:omics:us-west-2:123456789012:run/*",
        "arn:aws:omics:us-west-2:123456789012:runGroup/2345678"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "omics:GetRun",
        "omics:ListRunTasks",
        "omics:GetRunTask",
        "omics:CancelRun",
        "omics>DeleteRun"
      ],
      "Resource": [
        "arn:aws:omics:us-west-2:123456789012:run/*"
      ]
    },
  ]
}
```

}

## Using the CloudWatch Logs for troubleshooting

The CloudWatch Logs include run, task, and engine logs, which can be used to get updates on run progress or troubleshoot failed runs. The engine logs provide a detailed log of the data processing steps and analyses, and can be used to identify and correct errors. They are also useful for improving reproducibility and maintaining compliance with regulatory requirements.

### To view the CloudWatch Logs for workflows using the console

1. Open the HealthOmics console <https://console.aws.amazon.com/omics/>.
2. On the HealthOmics home page, choose  in the upper left corner of the screen to open the navigation pane. Select **Runs**.
3. Select the run from the runs list, which is organized by run ID.
4. When the run details page opens, choose **View Cloudwatch logs** to view the run logs. This links you to the CloudWatch console.
5. From the tasks page, select **View Logstream** to be linked to the engine logs for a further breakdown of errors.

Tasks logs can also be found in your AWS account log under the `/aws/omics/WorkflowLog` log group. Engine logs are only generated for failed workflow runs, and are organized in the log stream by run ID and engine, `run/{run-id}/task/{task-id}`.

# Cross-account resource sharing in AWS HealthOmics

Use cross-account sharing to share resources with collaborators without creating copies or modifying IAM resource policies. The following resources support cross-account sharing:

- HealthOmics variant stores
- HealthOmics annotation stores
- Private workflows

Sharing a resource includes the following steps:

1. The resource owner creates a share, and specifies the ARN of the resource and the AWS account of the intended subscriber. The resource share remains in pending state until the subscriber accepts the share.
2. The subscriber accepts the resource share to get access to the resource. The resource share transitions to activating state.
3. The HealthOmics service provides subscriber account with access to the resource.
4. The resource owner can delete the share, or the subscriber can revoke their access to the share. The subscriber can't delete the share or the associated resource.

## Topics

- [Create a share](#)
- [Retrieve information about a share](#)
- [View the shares that you own](#)
- [View accepted shares from other accounts](#)
- [Delete a share](#)

## Create a share

You can use the **create-share** API operation to create a share. The principal subscriber is the AWS account of the user who will subscribe to the shared resource. The following example creates a share for a variant store.

```
aws omics create-share \
```

```
--resource-arn "arn:aws:omics:us-west-2:555555555555:variantStore/omics_dev_var_store" \  
--principal-subscriber "123456789012" \  
--name "my_Share-123"
```

If the create is successful, you receive a response with the share ID and status.

```
{  
  "shareId": "495c21bedc889d07d0ab69d710a6841e-dd75ab7a1a9c384fa848b5bd8e5a7e0a",  
  "name": "my_Share-123",  
  "status": "PENDING"  
}
```

The share remains in **pending** state until the subscriber accepts it using the **accept-share** API operation.

```
aws omics accept-share \  
  --share-id "495c21bedc889d07d0ab69d710a6841e-dd75ab7a1a9c384fa848b5bd8e5a7e0a"
```

After the subscriber accepts the share, the share transitions to active state.

```
{  
  "status": "ACTIVATING"  
}
```

## Retrieve information about a share

Use the **get-share** API operation to retrieve information about the share.

```
aws omics get-share --share-id "495c21bedc889d07d0ab69d710a6841e-dd75ab7a1a9c384fa848b5bd8e5a7e0a"
```

The API response includes metadata information about the share.

```
{  
  "share":
```

```
{
  "shareId": "495c21bedc889d07d0ab69d710a6841e-dd75ab7a1a9c384fa848b5bd8e5a7e0a",
  "name": "my_Share-123",
  "resourceArn": "arn:aws:omics:us-west-2:555555555555:variantStore/
omics_dev_var_store",
  "principalSubscriber": "123456789012",
  "ownerId": "555555555555",
  "status": "PENDING"
}
```

## View the shares that you own

Use the **list-shares** API to retrieve information about each of the shares that you own.

```
aws omics list-shares --resource-owner SELF
```

The API response includes the metadata for each share that you own.

## View accepted shares from other accounts

Use the **list-shares** API to view all shares that you accepted from other accounts.

```
aws omics list-shares --resource-owner OTHER
```

The API response includes the metadata for each share that you accepted.

## Delete a share

Use the **delete-share** API to delete a share after you no longer need it.

```
aws omics delete-share \
  --share-id "495c21bedc889d07d0ab69d710a6841e-dd75ab7a1a9c384fa848b5bd8e5a7e0a"
```

# Tagging resources in HealthOmics

## Important notice

HealthOmics 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-inputted names for resources such as data stores or job-based operations are encrypted. They should never contain Personally Identifiable Information or Protected Health Information. For more information, see [Security in AWS HealthOmics](#).

## Tagging HealthOmics resources

You can assign metadata to your AWS resources using *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.

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 for a resource from the resource's service console, service API, or the AWS CLI.

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

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": "omics:Create*",
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": "omics:Start*",
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": "omics:Tag*",
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": "omics:Untag*",
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": "omics:List*",
    "Resource": "*"
  }
]
```

## Best practices

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

- Do not store Personally Identifiable Information (PII), Protected 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 don't 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 an HealthOmics resource

Adding tags to a resource 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 resource. You can use up to 50 tags per resource. There are also restrictions on the characters that you can use in the key and value fields.

After you add tags, you can create IAM policies to manage access to the AWS resource based on these tags. You can use the HealthOmics console or the AWS CLI to add tags to a resource. Adding tags to a repository can impact access to that repository. Before you add a tag to a data store, review any IAM policies that might use tags to control access to resources such as data stores.

Service tags are autogenerated for both a subject and a sample id for sequence stores.

Follow these steps to use the AWS CLI to add a tag to an HealthOmics resource. For example, to add tags to a sequence store while it's being created, you would use the following command in the



AWS CLI. The name of the sequence store is MySequenceStore, and the two added tags with keys are key1 and key2 with values as value1 and value2 respectively

:

```
aws omics create-sequence-store --name "MySequenceStore" --tags key1=value1,key2=value2
```

The output does not list the tags. It returns the following response.

```
{
  "id": "6860403586",
  "referenceStoreId": "4889894479",
  "roleArn": "arn:aws:iam::555555555555:role/ImportTest",
  "status": "CREATED",
  "creationTime": "2022-07-21T01:19:07.194Z"
}
```

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

```
aws omics tag-resource --resource-arn arn:aws:omics:us-west-2:555555555555:sequenceStore/2275234794 --tags key1=value1,key2=value2
```

If successful, this command returns no response.

## Listing tags for a resource

Follow these steps to use the AWS CLI to view a list of the AWS tags for an HealthOmics resource. 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 omics list-tags-for-resource --resource-arn arn:aws:omics:us-west-2:555555555555:sequenceStore/2275234794
```

You will receive a list of tags in response, in JSON format.

```
{
  "tags": {
```

```
    "key1": "value1",  
    "key2": "value2"  
  }  
}
```

## Removing tags from a data store

You can remove one or more tags associated with a resource. 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 resource where you want to remove tags and the tag key of the tag you want to remove.

```
aws omics untag-resource --resource-arn arn:aws:omics:us-  
west-2:555555555555:sequenceStore/2275234794 --tag-keys key1,key2
```

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

# AWS HealthOmics permissions

You can use AWS Identity and Access Management (IAM) to manage access to the HealthOmics API and resources such as stores and workflows. For users and applications in your account that use HealthOmics, you manage permissions in a permissions policy that you can apply to IAM users, groups, or roles.

To manage permissions for users and applications in your accounts, [use the policies that HealthOmics provides](#), or write your own. The HealthOmics console uses multiple services to get information about your function's configuration and triggers. You can use the provided policies as-is, or as a starting point for more restrictive policies.

HealthOmics uses IAM [service roles](#) to access other services on your behalf. For example, you would create or choose a service role when you run a workflow that reads data from Amazon S3. For some features, you also need to [configure permissions on resources in other services](#). Review these requirements before you start working with HealthOmics

For more information about IAM, see [What is IAM?](#) in the *IAM User Guide*.

## Topics

- [Identity-based IAM policies for HealthOmics](#)
- [Service roles for AWS HealthOmics](#)
- [Resource permissions](#)
- [Permissions for data access using Amazon S3 URIs](#)

## Identity-based IAM policies for HealthOmics

To grant users in your account access to HealthOmics, you use identity-based policies in AWS Identity and Access Management (IAM). Identity-based policies can apply directly to IAM users, or to IAM groups and roles that are associated with a user. You can also grant users in another account permission to assume a role in your account and access your HealthOmics resources.

The following IAM policy allows a user to access all HealthOmics API actions, and to pass [service roles](#) to HealthOmics.

### Example User policy

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "omics:*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": "omics.amazonaws.com"
      }
    }
  }
]
```

When you use HealthOmics, you also interact with other AWS services. To access these services, use the managed policies provided by each service. To restrict access to a subset of resources, you can use the managed policies as a starting point to create your own more restrictive policies.

- [AmazonS3FullAccess](#) – Access to Amazon S3 buckets and objects used by jobs.
- [AmazonEC2ContainerRegistryFullAccess](#) – Access to Amazon ECR registries and repositories for workflow container images.
- [AWSLakeFormationDataAdmin](#) – Access to Lake Formation databases and tables created by analytics stores.
- [ResourceGroupsandTagEditorFullAccess](#) – Tag HealthOmics resources with HealthOmics tagging API operations.

The preceding policies isn't allow a user to create IAM roles. For a user with these permissions to run a job, an administrator must create the service role that grants HealthOmics permission to access data sources. For more information, see [Service roles for AWS HealthOmics](#).

## Service roles for AWS HealthOmics

You can use service roles to grant AWS HealthOmics permission to access data and upload logs while processing a workflow or importing data to a Omics Storage or Omics Analytics data store. A service role is an AWS Identity and Access Management (IAM) role that an AWS service can use to access resources from other services in your account. You pass a service role to HealthOmics when you start a job.

Service roles must have the following trust policy.

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

The trust policy allows HealthOmics to assume the role.

### Sections

- [Sample IAM policies](#)
- [Sample CloudWatch templates](#)

## Sample IAM policies

The GitHub repository for this guide provides sample IAM policies that you can use as reference for developing service roles. You can use a single role that grants permission for both importing data and sending alerts by combining the applicable policies.

## Example Service role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "omics:*"
      ],
      "Resource": [
        "arn:aws:omics:us-west-2:123456789012:referenceStore/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:123456789012:log-group:/aws/omics/
WorkflowLog:log-stream:*"
      ]
    }
  ]
}
```

```

    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:123456789012:log-group:/aws/omics/
WorkflowLog:*"
      ]
    }
  ]
}

```

## Sample CloudWatch templates

The following sample template creates a service role that gives HealthOmics permission to access Amazon S3 buckets that have names prefixed with `omics-`, and to upload workflow logs.

### Example Reference store, Amazon S3 and CloudWatch Logs permissions

```

Parameters:
  bucketName:
    Description: Bucket name
    Type: String

Resources:
  serviceRole:
    Type: AWS::IAM::Role
    Properties:
      Policies:
        - PolicyName: read-reference
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action:
                  - omics:*
                Resource: !Sub arn:${AWS::Partition}:omics:${AWS::Region}:
${AWS::AccountId}:referenceStore/*
        - PolicyName: read-s3
          PolicyDocument:
            Version: 2012-10-17

```

```

Statement:
- Effect: Allow
  Action:
    - s3:ListBucket
  Resource: !Sub arn:${AWS::Partition}:s3:::${bucketName}
- Effect: Allow
  Action:
    - s3:GetObject
    - s3:PutObject
  Resource: !Sub arn:${AWS::Partition}:s3:::${bucketName}/*
- PolicyName: upload-logs
  PolicyDocument:
    Version: 2012-10-17
    Statement:
      - Effect: Allow
        Action:
          - logs:DescribeLogStreams
          - logs:CreateLogStream
          - logs:PutLogEvents
        Resource: !Sub arn:${AWS::Partition}:logs:${AWS::Region}:
${AWS::AccountId}:loggroup:/aws/omics/WorkflowLog:log-stream:*
      - Effect: Allow
        Action:
          - logs:CreateLogGroup
        Resource: !Sub arn:${AWS::Partition}:logs:${AWS::Region}:
${AWS::AccountId}:loggroup:/aws/omics/WorkflowLog:*
    AssumeRolePolicyDocument: |
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Action": [
              "sts:AssumeRole"
            ],
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "omics.amazonaws.com"
              ]
            }
          }
        ]
      }

```



# Resource permissions

AWS HealthOmics creates and accesses resources in other services on your behalf when you run a job or create a store. In some cases, you need to configure permissions in other services to access resources or to allow HealthOmics to access them.

## Sections

- [Lake Formation permissions](#)
- [Amazon ECR permissions](#)

## Lake Formation permissions

Before you use analytics features in HealthOmics, configure default database settings in Lake Formation.

### To configure resource permissions in Lake Formation

1. Open the [Data catalog settings](#) page in the Lake Formation console.
2. Uncheck the IAM access control requirements for databases and tables under **Default permissions for newly created databases and tables**.
3. Choose **Save**.

HealthOmics Analytics auto accepts data if your service policy has the correct RAM permissions, such as the following example.

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

```
        "ram:AcceptResourceShareInvitation",
        "ram:GetResourceShareInvitations"
    ],
    "Resource": "*"
}
]
```

## Amazon ECR permissions

When you run a workflow, the HealthOmics service accesses one or more containers that you provide in a private Amazon Elastic Container Registry (Amazon ECR). HealthOmics does not support access to public containers.

You create an IAM policy that grants HealthOmics access your private repository. You add this policy to each private repository that's referenced by a workflow. The private repository and workflow must be in the same region.

For additional details, see [Configure Amazon ECR permissions](#).

## Permissions for data access using Amazon S3 URIs

When you create a sequence store, HealthOmics adds permissions to the following methods in the creator's root account: `S3:GetObject`, `S3GetObjectTagging`, and `S3:ListBucket`. If HealthOmics owns the AWS KMS key on the sequence store, the root account also gets access to `kms:Decrypt`.

For a user in the account to access the data, you create a policy and attach it to the user or role to allow access to the files using Amazon S3 API operations. To use HealthOmics API operations, you must add HealthOmics permissions to your IAM policy. A policy allowing Amazon S3 API access can be applied at the sequence store level or at a read set level. At the read set level, permission can be restricted either through the prefix or using resource tag filters for sample or subject ID patterns.

The following example gives a user access to a sequence store. You can fine-tune the access with additional conditions or resource-based filters.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3DirectAccess",
```

```

    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket",
      "s3:GetObjectTagging"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "s3:DataAccessPointArn": "arn:aws:s3:us-
west-2:5555555555555555:accesspoint/592761533288-4891675750"
      }
    }
  },
  {
    "Sid": "DefaultSequenceStoreKMS",
    "Effect": "Allow",
    "Action": "kms.Decrypt",
    "Resource": "arn:aws:kms:us-west-2:5555555555555555:key/fa3b30f5-835d-4a6d-
b3f9-d3898fe0e648"
  }
]
}

```

To learn more about using IAM policies with HealthOmics, see [Service roles for AWS HealthOmics](#).

There are three ways you can use Amazon S3 URIs to share your data. The options are as follows:

- For sharing with users and roles within your account — Write a user access policy that includes access to the AWS KMS key and access to the access point. This makes the data accessible for use with the Amazon S3 API operations.
- For sharing with users outside of your account — Create a role within the data owner's account that has an access policy that allows the user to assume that role. Adding the user with direct access isn't supported.
- Presigned URLs — You can also generate a shareable URL for a file in the sequence store.

To learn more about creating presigned URLs by using Amazon S3, see [Using presigned URL](#) in the Amazon S3 documentation.

# Security in AWS HealthOmics

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are 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 AWS HealthOmics, 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 AWS HealthOmics. The following topics show you how to configure AWS HealthOmics to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS HealthOmics resources.

## Topics

- [Data protection in AWS HealthOmics](#)
- [Identity and access management for HealthOmics](#)
- [Compliance validation for AWS HealthOmics](#)
- [Resilience in HealthOmics](#)
- [AWS HealthOmics and interface VPC endpoints \(AWS PrivateLink\)](#)

## Data protection in AWS HealthOmics

The AWS [shared responsibility model](#) applies to data protection in AWS HealthOmics. 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 AWS HealthOmics 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

AWS HealthOmics 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. To learn more about Customer-managed KMS Key, see [Amazon Key Management Service](#).

All HealthOmics data stores (Storage and Analytics) support the use of Customer-managed KMS keys. 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.

For HealthOmics Workflows, customer-managed keys aren't supported by the temporary file system; however, all data is encrypted at rest automatically using XTS-AES-256 block cipher encryption algorithm to encrypt the file system. The IAM user and role used to start a workflow run must also have access to the AWS KMS keys used for workflow input and output buckets. Workflows does not use grants, and AWS KMS encryption is limited to input and output Amazon S3 buckets. The IAM role used both for workflow APIs must also have access to the AWS KMS keys used as well as the input and output Amazon S3 buckets. You can use either IAM roles and permissions to control access or AWS KMS policies. To learn more, see [Authentication and access control for AWS KMS](#).

Additionally, when using AWS Lake Formation with HealthOmics Analytics, any decrypt permissions associated with the Lake Formation are also given to the input and output Amazon S3 buckets. More information about how AWS Lake Formation manages permissions can be found in the [AWS Lake Formation documentation](#).

HealthOmics Analytics grants Lake Formation `kms:Decrypt` permissions to read the encrypted data in an Amazon S3 bucket. As long as you have permissions to query the data through Lake Formation, you will be able to read the encrypted data. Access to the data is controlled through data access control in Lake Formation, not through a KMS key policy. To learn more, see the [AWS Integrated AWS service requests](#) in the Lake Formation documentation.

## AWS owned KMS key

AWS HealthOmics uses these keys by default to automatically encrypt potentially sensitive information such as personally identifiable or Protected 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

AWS HealthOmics 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 AWS HealthOmics 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 AWS HealthOmics resources, [kms:CreateGrant](#) operations must be permitted in the key policy. This adds a grant to a customer managed key that controls access to a specified KMS key. This key gives a user access to the [kms:grant operations](#) that AWS HealthOmics requires. See [Using grants](#) for more information.

To use your customer managed KMS key with your AWS HealthOmics 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 in HealthOmics Analytics and HealthOmics Storage. HealthOmics Workflows does not use grants.
- 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 role to create and interact with a data store in AWS HealthOmics which is encrypted by that key:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "omics.amazonaws.com"
      },
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:Encrypt",
        "kms:GenerateDataKey*",
        "kms:ReEncrypt*"
      ],
      "Resource": "*"
    }
  ]
}
```

The following policy would create permissions for a data store to decrypt data from an Amazon S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "omics:GetReference",

```



```

        "omics:GetReferenceMetadata"
    ],
    "Resource": [
        "arn:AWS:omics:{{region}}:{{accountId}}:referenceStore/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject"
    ],
    "Resource": [
        "arn:AWS:s3:::[s3path]/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt"
    ],
    "Resource": [
        "arn:AWS:kms:{{region}}:{{account_id}}:key/{{key_id}}"
    ]
    "Condition": {
        "StringEquals": {
            "kms:ViaService": [
                "s3.{{region}}.amazonAWS.com"
            ]
        }
    }
}
]
}

```

## Required IAM permissions for using a customer managed KMS key

When creating a resource such as a data store with AWS KMS encryption using a customer managed KMS key, there are required permissions for both the key policy and the IAM policy for the IAM user or role.

You can use the [kms:ViaService condition key](#) to limit use of the KMS key to only requests that originate from AWS HealthOmics.

For more information about key policies, see [Enabling IAM policies](#) in the AWS Key Management Service Developer Guide.

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

## How AWS HealthOmics uses grants in AWS KMS

HealthOmics Analytics requires a [grant](#) to use your customer managed KMS key. Grants aren't required or used for either HealthOmics Workflows or HealthOmics Storage. When you create a data store encrypted with a customer managed KMS key, AWS HealthOmics creates a grant on your behalf by sending a [CreateGrant](#) request to AWS KMS. Grants in AWS KMS are used to give AWS HealthOmics access to a KMS key in a customer account.

It isn't recommended to revoke or retire the grants that AWS HealthOmics creates on your behalf. If you revoke or retire the grant that gives AWS HealthOmics permission to use the AWS KMS keys in your account, AWS HealthOmics cannot access this data, encrypt new resources pushed to the data store, or decrypt them when they are pulled. When you revoke or retire a grant for AWS HealthOmics, 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, AWS HealthOmics retires the grants on your behalf.

## Monitoring your encryption keys for AWS HealthOmics

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

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

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

You aren't required to create your own grants. AWS HealthOmics creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give AWS HealthOmics access to a AWS KMS key in a customer account.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "xx:test",
    "arn": "arn:AWS:sts::555555555555:assumed-role/user-admin/test",
    "accountId": "xx",
    "accessKeyId": "xxx",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "xxxx",
        "arn": "arn:AWS:iam::555555555555:role/user-admin",
        "accountId": "555555555555",
        "userName": "user-admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-11-11T01:36:17Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "invokedBy": "apigateway.amazonAWS.com"
},
"eventTime": "2022-11-11T02:34:41Z",
"eventSource": "kms.amazonAWS.com",
"eventName": "CreateGrant",
"AWSRegion": "us-west-2",
"sourceIPAddress": "apigateway.amazonAWS.com",
"userAgent": "apigateway.amazonAWS.com",
"requestParameters": {
  "granteePrincipal": "AWS Internal",
  "keyId": "arn:AWS:kms:us-west-2:555555555555:key/a6e87d77-cc3e-4a98-a354-
e4c275d775ef",
  "operations": [
    "CreateGrant",
    "RetireGrant",
    "Decrypt",
    "GenerateDataKey"
  ]
}
```

```

    ]
  },
  "responseElements": {
    "grantId": "4869b81e0e1db234342842af9f5531d692a76edaff03e94f4645d493f4620ed7",
    "keyId": "arn:AWS:kms:us-west-2:245126421963:key/xx-cc3e-4a98-a354-
e4c275d775ef"
  },
  "requestID": "d31d23d6-b6ce-41b3-bbca-6e0757f7c59a",
  "eventID": "3a746636-20ef-426b-861f-e77efc56e23c",
  "readOnly": false,
  "resources": [
    {
      "accountId": "245126421963",
      "type": "AWS::KMS::Key",
      "ARN": "arn:AWS:kms:us-west-2:245126421963:key/xx-cc3e-4a98-a354-
e4c275d775ef"
    }
  ],
  "eventType": "AWSApiCall",
  "managementEvent": true,
  "recipientAccountId": "245126421963",
  "eventCategory": "Management"
}

```

The following example shows how to use `GenerateDataKey` to ensure the user has the 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": "omics.amazonAWS.com"
},
"eventTime": "2021-06-30T21:17:37Z",
"eventSource": "kms.amazonAWS.com",
"eventName": "GenerateDataKey",
"AWSRegion": "us-east-1",
"sourceIPAddress": "omics.amazonAWS.com",
"userAgent": "omics.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"
}
```

## 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

AWS HealthOmics uses TLS 1.2+ to encrypt data in transit through the public endpoints and through backend services.

## Identity and access management for HealthOmics

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 AWS HealthOmics resources. IAM is an AWS service that you can use with no additional charge.

### Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How AWS HealthOmics works with IAM](#)
- [Identity-based policy examples for AWS HealthOmics](#)
- [AWS managed policies for AWS HealthOmics](#)
- [Troubleshooting AWS HealthOmics identity and access](#)

## Audience

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

**Service user** – If you use the AWS HealthOmics service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS HealthOmics 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 AWS HealthOmics, see [Troubleshooting AWS HealthOmics identity and access](#).

**Service administrator** – If you're in charge of AWS HealthOmics resources at your company, you probably have full access to AWS HealthOmics. It's your job to determine which AWS HealthOmics

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 AWS HealthOmics, see [How AWS HealthOmics 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 AWS HealthOmics. To view example AWS HealthOmics identity-based policies that you can use in IAM, see [Identity-based policy examples for AWS HealthOmics](#).

## 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 HealthOmics works with IAM

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

### IAM features you can use with AWS HealthOmics

IAM feature	HealthOmics support
<a href="#">Identity-based policies</a>	Yes
<a href="#">Resource-based policies</a>	No
<a href="#">Policy actions</a>	Yes
<a href="#">Policy resources</a>	Yes
<a href="#">Policy condition keys</a>	No
<a href="#">ACLs</a>	No
<a href="#">ABAC (tags in policies)</a>	Yes
<a href="#">Temporary credentials</a>	Yes
<a href="#">Principal permissions</a>	Yes
<a href="#">Service roles</a>	Yes
<a href="#">Service-linked roles</a>	No

To get a high-level view of how HealthOmics and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

## Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it shouldn't otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the [aws:SourceArn](#) and [aws:SourceAccount](#) global condition context keys in resource policies to limit the permissions that AWS HealthOmics gives another service to the resource.

To prevent the confused deputy problem in roles assumed by HealthOmics, set the value of `aws:SourceArn` to `arn:aws:omics:region:accountNumber:*` in the role's trust policy. The wildcard (\*) applies the condition for all HealthOmics resources.

The following trust relationship policy grants HealthOmics access to your resources and uses the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to prevent the confused deputy problem. Use this policy when you create a role for HealthOmics.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "omics.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "accountNumber"
        },
        "StringLike": {
```

```
        "aws:SourceArn": "arn:aws:omics:region:accountNumber:*"
    }
}
]
```

## Identity-based policies for HealthOmics

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 HealthOmics

To view examples of AWS HealthOmics identity-based policies, see [Identity-based policy examples for AWS HealthOmics](#).

## Resource-based policies within HealthOmics

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 HealthOmics

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 HealthOmics actions, see [Actions Defined by AWS HealthOmics](#) in the *Service Authorization Reference*.

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

```
healthomics
```

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

```
"Action": [
```



```
"healthomics:action1",  
"healthomics:action2"  
]
```

To view examples of AWS HealthOmics identity-based policies, see [Identity-based policy examples for AWS HealthOmics](#).

## Policy resources for HealthOmics

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 HealthOmics resource types and their ARNs, see [Resources Defined by AWS HealthOmics](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by AWS HealthOmics](#).

To view examples of AWS HealthOmics identity-based policies, see [Identity-based policy examples for AWS HealthOmics](#).

## Policy condition keys for HealthOmics

Policy condition keys aren't supported in HealthOmics.

## Access control lists (ACLs) in HealthOmics

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 HealthOmics

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*.

For more information about tagging HealthOmics resources, see [Tagging resources in HealthOmics](#).

The following example shows how you can write an IAM policy denying access to a resource without a specific tag.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "omics:*"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "Null": {
          "aws:RequestTag/MyCustomTag": "true"
        }
      }
    }
  ]
}
```

You can also limit access to a runs within a run group, as shown.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "omics:StartRun"
      ],
      "Resource": [
        "arn:aws:omics:us-west-2:123456789012:run/*",
        "arn:aws:omics:us-west-2:123456789012:workflow/1234567",
        "arn:aws:omics:us-west-2:123456789012:runGroup/2345678"
      ],
      "Condition": {
        "StringLike": {
          "omics:Workflow": "arn:aws:omics:us-east-1:123456789012:workflow/*"
        },
        "StringLike": {
```

```
        "omics:RunGroup": "arn:aws:omics:us-east-1:123456789012:runGroup/*"
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "omics:GetRun",
        "omics:ListRunTasks",
        "omics:GetRunTask",
        "omics:CancelRun",
        "omics>DeleteRun"
      ],
      "Resource": [
        "arn:aws:omics:us-west-2:123456789012:run/*"
      ]
    },
  ]
}
```

## Using Temporary credentials with HealthOmics

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 HealthOmics

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 HealthOmics

Supports service roles	Yes
------------------------	-----

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

### Warning

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

## Service-linked roles for HealthOmics

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 HealthOmics

By default, users and roles don't have permission to create or modify AWS HealthOmics 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 AWS HealthOmics, including the format of the ARNs for each of the resource types, see [Actions, Resources, and Condition Keys for AWS HealthOmics](#) in the *Service Authorization Reference*.

### Topics

- [Policy best practices](#)
- [Using the HealthOmics console](#)
- [Allow users to view their own permissions](#)

## Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS HealthOmics 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 HealthOmics console

To access the AWS HealthOmics console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS HealthOmics 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.

## Allow users to view their own permissions

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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 HealthOmics



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: AmazonOmicsFullAccess**

You can attach the AmazonOmicsFullAccess policy to your IAM identities to give them full access to HealthOmics.

This policy grants full access permissions to all HealthOmics actions. When you create an annotation or variant store, Omics will also give you access to those stores through a Resource Share Invitation in the Resource Access Manager (RAM) console. For more information on Resource Share invitations through Lake Formation, see the [Lake Formation documentation](#). For an Omics admin policy, you will also need the following permissions to access your Amazon S3 bucket.

- PutObject
- GetObject
- ListBucket
- AbortMultipartUpload

- ListMultipartUploadParts

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "omics:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ram:AcceptResourceShareInvitation",
        "ram:GetResourceShareInvitations"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "omics.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "omics.amazonaws.com"
        }
      }
    }
  ]
}
```

## AWS managed policy: AmazonOmicsReadOnlyAccess

You can attach the `AWSOmicsReadOnlyAccess` policy to your IAM identities when you wish to limit the permissions for that identity to read-only access.

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

## HealthOmics updates to AWS managed policies

View details about updates to AWS managed policies for HealthOmics since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the HealthOmics Document history page.

Change	Description	Date
AmazonOmicsFullAccess - New policy added	HealthOmics added a new policy to grant a user full access to all actions and resources. To learn more, see <a href="#">AmazonOmicsFullAccess</a> .	February 23, 2023
HealthOmics started tracking changes	HealthOmics started tracking changes for its AWS managed policies.	November 29, 2022

Change	Description	Date
AmazonOmicsReadOnlyAccess - New policy added	HealthOmics added a new policy that limits access to read only. To learn more, <a href="#">AmazonOmicsReadOnlyAccess</a> .	November 29, 2022

## Troubleshooting AWS HealthOmics identity and access

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

### Topics

- [I am not authorized to perform an action in HealthOmics](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my HealthOmics resources](#)

### I am not authorized to perform an action in HealthOmics

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

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 doesn't have the fictional healthomics:*GetWidget* permissions.

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

In this case, the policy for the mateojackson user must be updated to allow access to the *my-example-widget* resource by using the healthomics:*GetWidget* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

## 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 AWS HealthOmics.

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 AWS HealthOmics. 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 HealthOmics 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 AWS HealthOmics supports these features, see [How AWS HealthOmics 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*.

## Compliance validation for AWS HealthOmics

Third-party auditors assess the security and compliance of AWS HealthOmics as part of multiple AWS compliance programs. This includes HIPAA, FedRAMP, and others.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

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

Your compliance responsibility when using AWS HealthOmics 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 security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

## Resilience in HealthOmics

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, AWS HealthOmics offers several features to help support your data resiliency and backup needs.

## AWS HealthOmics and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and AWS HealthOmics by creating an *interface VPC endpoint*. Interface endpoints are powered by [AWS PrivateLink](#), a technology that you can use to privately access HealthOmics API operations without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't require public IP addresses to communicate with HealthOmics API operations. Traffic between your VPC and HealthOmics doesn't go outside 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*.

VPC endpoint policies are supported for HealthOmics for all Regions except Israel (Tel Aviv). By default, full access to HealthOmics is allowed through the endpoint.

## Considerations for HealthOmics VPC endpoints

Before you set up an interface VPC endpoint for HealthOmics, make sure that you review [Interface endpoint properties and limitations](#) in the *Amazon VPC User Guide*.

HealthOmics supports making calls to all HealthOmics Storage API actions from your VPC.

VPC endpoint policies aren't supported for HealthOmics by default, but you can create a VPC endpoint for full HealthOmics access for the HealthOmics Storage operations. For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

## Creating an interface VPC endpoint for HealthOmics

You can create a VPC endpoint for the HealthOmics service by using 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 HealthOmics by using the following service names:

- `com.amazonaws.region.storage-omics`
- `com.amazonaws.region.control-storage-omics`
- `com.amazonaws.region.analytics-omics`
- `com.amazonaws.region.workflows-omics`
- `com.amazonaws.region.tags-omics`

If you turn on private DNS for the endpoint, you can make API requests to HealthOmics by using its default DNS name for the Region, for example, `omics.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 HealthOmics

You can attach an endpoint policy to your VPC endpoint that controls access to HealthOmics. 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 HealthOmics actions.



The following is an example of an endpoint policy for HealthOmics. When attached to an endpoint, this policy grants access to HealthOmics actions for all principals on all resources.

## API

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "omics:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

## AWS CLI

```
aws ec2 modify-vpc-endpoint \
  --vpc-endpoint-id vpce-id \
  --region us-west-2 \
  --policy-document \
  "{\"Statement\": [{\"Principal\": \"*\", \"Effect\": \"Allow\", \"Action\": \"omics:List*\"}, {\"Resource\": \"*\"}]}"
```

## Special considerations for accessing read sets using Amazon S3 URIs

To access read sets through Amazon S3 URIs when you're using a private connection, set up the PrivateLink interface endpoints on the sequence store. After you set them up, the endpoints have the following formats:

```
com.amazonaws.region.storage-omics
com.amazonaws.region.control-storage-omics
```

To use Gateway endpoints, follow the guide [Gateway endpoints for Amazon S3](#) to configure your gateway endpoints. HealthOmics owns the Amazon S3 bucket, so you don't have to create or adjust the bucket policy. Gateway endpoints rely on the policy attached to the user or role that accesses

the data, but you can also configure endpoints with more restrictive policies. These policies can include restrictions on access based on the Amazon S3 Access Point ARN and Amazon S3 actions.

# Monitoring AWS HealthOmics

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

- *Amazon CloudWatch* monitors your AWS resources 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 threshold that you specify. 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](#).
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and 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 from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).

*Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see the [Amazon EventBridge User Guide](#).

## Topics

- [Monitoring HealthOmics with Amazon CloudWatch](#)
- [Logging AWS HealthOmics API calls using AWS CloudTrail](#)
- [Using EventBridge with AWS HealthOmics](#)

# Monitoring HealthOmics with Amazon CloudWatch

You can monitor HealthOmics using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access 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](#).

The AWS HealthOmics service reports the following metrics in the `AWS/Omics` namespace.

API Call Count metrics are reported for the following AWS HealthOmics APIs. Only the API Operation dimension is reported.

- Reference and reference store APIs — `CreateReferenceStore`, `DeleteReferenceStore`, `StartReferenceImportJob`
- Sequence store and read set APIs — `CreateSequenceStore`, `DeleteSequenceStore`, `StartReadSetImportJob`, `StartReadSetActivationJob`, `StartReadSetExportJob`
- Variant store APIs — `CreateVariantStore`, `DeleteVariantStore`, `StartVariantImportJob`, `CancelVariantImportJob`
- Annotation store APIs — `CreateAnnotationStore`, `DeleteAnnotationStore`, `StartAnnotationImportJob`, `CancelAnnotationImportJob`
- Workflow, run, and run group APIs — `CreateWorkflow`, `DeleteWorkflow`, `StartRun`, `CancelRun`, `DeleteRun`, `CreateRunGroup`, `DeleteRunGroup`

## Viewing *AWS HealthOmics* metrics

CloudWatch metrics for AWS HealthOmics are viewable in the CloudWatch console.

### 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/Usage**.
3. Filter **Service** for **AWS HealthOmics**.
4. Choose the dimension, choose a metric name, then choose **Add to graph**.
5. 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/Usage**, and then choose an AWS HealthOmics metric using the Service dimension.
4. For **Time Range**, choose a time range to monitor, and then choose **Next**.
5. Enter a **Name** and **Description**.
6. For **Whenever**, choose  $\geq$ , 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**.

## Logging AWS HealthOmics API calls using AWS CloudTrail

AWS HealthOmics is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in HealthOmics. CloudTrail captures all API calls for HealthOmics as events. The calls captured include calls from the HealthOmics console and code calls to the HealthOmics API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for HealthOmics. 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 HealthOmics, 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](#).

## HealthOmics information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in HealthOmics, 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 HealthOmics, 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 HealthOmics actions are logged by CloudTrail and are documented in the [AWS HealthOmics API Reference](#). For example, calls to the `CreateReferenceStore`, `StartVariantImportJob` and `CreateWorkflow` actions generate entries in the CloudTrail log files.

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 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 HealthOmics 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 CreateWorkflow action.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAIU53LOGOMTOPXXNPG:username",
    "arn": "arn:aws:sts::account:assumed-role/admin/username",
    "accountId": "account-id",
    "accessKeyId": "accessKeyId",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAIU53LOGOMTOPXXNPG",
        "arn": "arn:aws:iam::account:role/admin",
        "accountId": "account",
        "userName": "admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-07-23T18:26:09Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2022-07-23T18:46:42Z",
  "eventSource": "omics.amazonaws.com",
  "eventName": "CreateWorkflow",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "205.251.233.176",
  "userAgent": "aws-cli/1.22.45 Python/3.9.13 Darwin/20.6.0 boto3/1.23.45",
  "requestParameters": {
    "name": "parameter_name",
    "definitionZip": "czM6Ly93b3JrZmxvd2RlZi1oZWxsby9kZWZpbml0aW9uLnppcA==",
    "requestId": "d788a73c-b81b-45fb-a8a6-d8bb4449ec8a"
  }
}
```

```

    },
    "responseElements": {
      "id": "1002571",
      "arn": "arn:aws:omics:us-west-2:555555555555:instance/i-b188560f ",
      "status": "CREATING",
      "tags": {
        "resourceArn": "arn:aws:omics:us-west-2:083685709690:workflow/1002571"
      }
    },
    "requestID": "842d731d-f264-4b08-a2c9-2f7d45e1eaa3",
    "eventID": "76872ca2-f208-4193-807d-7dd7ea34e6b2",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "083685709690",
    "eventCategory": "Management"
  }
}

```

## Using EventBridge with AWS HealthOmics

HealthOmics generates and sends events to Amazon EventBridge as a best effort delivery to your account's default bus. After EventBridge is enabled, all events are sent to EventBridge. You can use EventBridge rules to route events to additional targets. The following table lists the events that are sent to EventBridge.

Resource or event	Possible status description
Variant or annotation store	Creating, created, updating, updated, deleting, deleted, or creation failed
Reference or sequence store	Created or deleted
Read sets	Processing upload, upload failed, active, archived, activating, or deleted
Variant or annotation import job	Submitted, in progress, cancelled, completed, failed, or completed with failures
Reference import or export job, read set import or export	Submitted, in progress, completed, failed, or completed with failures



Resource or event	Possible status description
Read set activation job	Submitted, in progress, completed, failed, or completed with failures
Workflow status	Possible statuses are creation success, creation failure, deletion success, or deletion failure
Run group status	Active or deleted
Run or task	Pending, starting, running, stopping, completed, deleted, failed, or cancelled
Reference	Active or deleted

## EventBridge event message structure and examples

HealthOmics sends events to Amazon EventBridge whenever a resource is created, updated, deleted, or changes state. You can use EventBridge and these events to write rules that take actions, such as notifying you when a resource changes state. For more information, see [What is Amazon EventBridge?](#)

HealthOmics provides best effort delivery of state changes to EventBridge. The event is an object with JSON structure that also contains metadata details. You can use the metadata as input to either recreate the event or learn more information. The following fields are included:

- `version` — Currently 0 (zero) for all events.
- `id` — A Version 4 UUID generated for every event.
- `detail-type` — The type of event that's being sent.
- `account` — The 12-digit AWS account ID of the bucket owner.
- `source` — Identifies the service that generated the event.
- `time` — The time the event occurred.
- `region` — Identifies the AWS Region of the bucket.
- `resources` — A JSON array that contains the Amazon Resource Name (ARN) of the bucket.
- `detail` — A JSON object that contains information about the event.

For run events, the following fields are included:

- `uuid` — The universally unique identifier for the run.
- `runOutputUri` — The URI for where the run will write its output data.

The following is an example of an event that's created when a read set status changes.

```
{
  "version": "0"
  "id": "64ca0eda-9751-dc55-c41a-1bd50b4fc9b7",
  "detail-type": "Read Set Status Change",
  "source": "aws.omics",
  "account": "123456789012",
  "time": "2023-04-04T17:53:06Z",
  "region": "us-west-2",
  "resources": ["arn:aws:omics:us-west-2:123456789012:sequenceStore/1234567890/readSet/3456789012"],
  "detail": {
    "omicsVersion": "1.0.0",
    "arn": "arn:aws:omics:us-west-2:123456789012:sequenceStore/1234567890/readSet/3456789012",
    "sequenceStoreId" : "1234567890",
    "id": "3456789012",
    "status": "PROCESSING_UPLOAD"
  }
}
```

A similar event gets created for a variant store import job.

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "Variant Store Status Change",
  "source": "aws.omics",
  "account": "123456789012",
  "time": "2015-12-22T18:43:48Z",
  "region": "us-east-1",
  "resources": ["arn:aws:omics:us-east-1:123456789012:bcvariantstore2"],
  "detail": {
    "omicsVersion": "1.0.0",
    "arn": "arn:aws:omics:us-east-1:123456789012:bcvariantstore2",
    "status": "CREATED",
  }
}
```

```

    "storeId": "6710c5f02610",
    "storeName": "bcvariantstore2",
  }
}

```

The following is an event that would be created for a change in import job status.

```

{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "Variant Import Job Status Change",
  "source": "aws.omics",
  "account": "123456789012",
  "time": "2015-12-22T18:43:48Z",
  "region": "us-east-1",
  "resources": ["arn:aws:omics:us-
east-1:123456789012:vincent_load_test_variant_store/
b64ea9a3-459f-4b68-92c3-3ddb83209fe9"],
  "detail": {
    "omicsVersion": "1.0.0",
    "arn": "arn:aws:omics:us-east-1:123456789012:vincent_load_test_variant_store/
b64ea9a3-459f-4b68-92c3-3ddb83209fe9",
    "status": "COMPLETED",
    "jobId": "b64ea9a3-459f-4b68-92c3-3ddb83209fe9",
    "storeId": "a74869f91e20",
    "storeName": "vincent_load_test_variant_store"
  }
}

```

For run events, two additional fields are included in the detail field. These are the `uuid` and `runOutputUri`, as shown in the following example.

```

{
  "version": "0",
  "id": "c0e540f4-df38-b986-86c1-3e3730f971fe",
  "detail-type": "Run Status Change",
  "source": "aws.omics",
  "account": "123456789012",
  "time": "2022-10-20T22:07:35Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:omics:us-west-2:123456789012:run/2101313"
  ]
}

```

```
],  
  "detail":{  
    "omicsVersion":"1.0.0",  
    "arn":"arn:aws:omics:us-west-2:123456789012:run/2101313",  
    "status":"COMPLETED",  
    "uuid":"153893cd-097a-40ec-aec7-838a97cd2b21",  
    "runOutputUri":"s3://integ-test-0ee27e7e/run-output/2101313"  
  }  
}
```

# Troubleshooting

The following documentation can help you troubleshoot problems you might have with your HealthOmics data stores and workflows.

## Topics

- [Why can't I run my workflow?](#)
- [Why do I get a "not a currently supported operation" error when running Nextflow?](#)
- [Why can't I create a reference store?](#)
- [Why can't I create a sequence store?](#)
- [Why can't I create a workflow?](#)
- [Why did my task fail?](#)
- [Why can't I import my BAM, CRAM or FASTQ files?](#)
- [Why can't I import my VCF or gVCF files?](#)
- [Why can't I see my annotation store or variant store in Athena?](#)
- [Why can't I access my data store in Athena?](#)
- [Why do I get a "Request Too Long" error message when I try to create a workflow?](#)
- [Error and status messages for run failures](#)

## Why can't I run my workflow?

If you're getting an error to run your workflow while using the AWS CLI, check to make sure you have the latest version installed. Also make sure you have the correct Amazon ECR and IAM permissions to access your input data.

A full log of failed tasks can be found in your engine logs, which can be accessed either through the console or your AWS account. Tasks logs can also be found in your AWS account log under the `/aws/omics/WorkflowLog` log group. Engine logs are only generated for failed workflow runs, and are organized in the log stream by Run ID and engine, `run/{run-id}/task/{task-id}`.

## Why do I get a "not a currently supported operation" error when running Nextflow?

If you receive the following error when running Nextflow:

```
java.lang.UnsupportedOperationException:  
    BaseFileSystemProvider.createDirectory is not a currently supported operation
```

Check that your Nextflow definition file sets the **publishDir** value to `/mnt/workflow/pubdir`. To export files to Amazon S3, the files must be in this directory. For more information, see the Nextflow example in [Workflow definition file examples](#).

## Why can't I create a reference store?

If you receive an error that states "You don't have permissions for this action with the credentials you sent," check your IAM permissions. Confirm you have permission in your policy for the role you are using to use the `CreateReferenceStore` action.

## Why can't I create a sequence store?

If you receive an error that states "You don't have permissions for this action with the credentials you sent," check your IAM permissions. Confirm you have permission in your policy for the role you are using to use the `CreateSequenceStore` action.

## Why can't I create a workflow?

If the error states "Zip file contains multiple workflow definition files", you must specify the main/master WDL or Nextflow file to use. You can do this by entering in the main path to the file as part of the command `--main folder_name/file_name` through the AWS CLI. You can also do this using the main workflow's definition file field for the console.

## Why did my task fail?

Make sure that you have the appropriate service role. This role must be able to read from the Amazon Simple Storage Service location(s) or sequence store where your data resides. It must

also have the appropriate trust policy for HealthOmics to assume the role. You can find more information in the CloudWatch Logs.

## Why can't I import my BAM, CRAM or FASTQ files?

Make sure that you have the appropriate service role. This role must be able to read from any Amazon S3 location where your data resides. If you are using a customer managed key(CM-CMK), you must also use the AWS KMS decrypt permissions. It must also have the appropriate trust policy for HealthOmics to assume the role.

## Why can't I import my VCF or gVCF files?

Make sure that you have the appropriate service role. This role must be able to read from any Amazon S3 location where your data resides. If you are using a customer managed key(CM-CMK), you must also use the AWS KMS decrypt permissions. It must also have the appropriate trust policy for HealthOmics to assume the role.

## Why can't I see my annotation store or variant store in Athena?

In Lake Formation, be sure to create a resource link based on the store that was shared with you. Once you create a resource link that you have permission to access, the store should be visible in Athena.

## Why can't I access my data store in Athena?

If your annotation or variant store is visible but you are receiving an error message saying that access is denied, check which query engine version you're using. Only queries run using engine version 3 are supported. To read more about Athena query engine versions, see the [Amazon Athena documentation](#).

## Why do I get a "Request Too Long" error message when I try to create a workflow?

If you're creating a workflow with a definition zip embedded in the request, you may get a "Request Too Long" error message. You can work around this issue by uploading the definition zip to an Amazon S3 bucket.

# Error and status messages for run failures

To learn more about why a run a has failed, use the **GetRun** API operation.

To troubleshoot a run failure, refer to the following error codes and statuses. This table lists failures and messages.

## Service error messages

Failure reason	Detailed error description
INSTANCE_RESERVATION_FAILED	There isn't enough instance capacity to complete the workflow run. Wait and try the workflow run again.
IMPORT_FAILED	The workflow run didn't finish because of a transient error while importing <i>uri</i> . Try the workflow run again.
EXPORT_FAILED	The workflow run didn't finish because of a transient error while exporting the run output. Try the workflow run again.
INVALID_URI_INPUT	The URI structure isn't a valid <i>uri</i> . Check the URI structure and try again.
INVALID_S3_INPUT	The URI does not exist: <i>uri</i> . Check that the URI path exists and confirm that the role can access the object.
INVALID_OMICS_STORAGE_INPUT	The HealthOmics storage URI does not exist: <i>uri</i> . Check that the read set path exists and confirm that the role can access the read set.
INACTIVE_OMICS_STORAGE_RESOURCE	The HealthOmics storage URI isn't in ACTIVE state. Activate the read set and try again. To learn more about activating read sets, see <a href="#">Activating read sets</a> .



Failure reason	Detailed error description
MODIFIED_INPUT_RESOURCE	The input URI was modified after the run started.
ECR_PERMISSION_ERROR	HealthOmics doesn't have permission to access the image URI. Confirm that the Amazon ECR private repository exists and has granted access to the HealthOmics service principal.
INVALID_ECR_IMAGE_URI	The Amazon ECR image URI structure isn't valid. Check for a valid URI and try again.
INVALID_TASK_RESOURCE_VALUE	The requested GPU, CPU, or memory is either too high for available compute capacity, or is less than the minimum value of 1 for task <i>ID</i> .
OUT_OF_MEMORY_ERROR	The workflow task <i>ID</i> ran out of memory. Increase the memory value in the workflow definition and try the run again.
RUN_TASK_FAILED	The workflow run failed because the task failed. To debug task failure, use the <b>GetRunTask</b> API operation and the Amazon CloudWatch Logs stream.
IMPORT_FAILED	The import failed. Check that the input file exists and the run role can access input.
EXPORT_FAILED	The export failed. Check that the output bucket exists and the run role has write permission to the bucket.
ASSUME_ROLE_FAILED	HealthOmics doesn't have permission to assume the role. Specify the HealthOmics principal in the trust relationship for the role.

<b>Failure reason</b>	<b>Detailed error description</b>
UNSUPPORTED_INPUT_SIZE	The total input size is too high. Decrease the input size and try again.
SERVICE_ERROR	The workflow run didn't finish because of a transient service error. Try the workflow run again.

# Quotas for AWS HealthOmics

AWS initially populates your account with default values for the HealthOmics service quotas. Unless otherwise noted, each quota value is the per-Region maximum value. You can request increases for some quotas, and other quotas have fixed values (not adjustable).

## Topics

- [HealthOmics service quotas](#)
- [HealthOmics file size quotas](#)
- [HealthOmics API quotas](#)

## HealthOmics service quotas

The table below lists the HealthOmics service quotas, along with their default values.

To view the current quotas for each Region, open the [Service Quotas console](#). In the navigation pane, choose **AWS services** and select **HealthOmics**.

To request a quota increase, see [Requesting a Quota Increase](#) in the *Service Quotas User Guide*. If the quota isn't yet available in Service Quotas, use the [quota increase form](#).

Name	Default	Adjustable	Description
Analytics - Maximum annotation stores	Each supported Region: 10	<a href="#">Yes</a>	The maximum number of annotation stores in the current AWS region
Analytics - Maximum concurrent variant or annotation store import jobs	Each supported Region: 5	<a href="#">Yes</a>	The maximum number of concurrent import jobs in the current AWS region
Analytics - Maximum files per annotation store import job	Each supported Region: 1	No	The maximum number of files per annotation import job in the current AWS region

Name	Default	Adjustable	Description
Analytics - Maximum files per variant store import job	Each supported Region: 1,000	<a href="#">Yes</a>	The maximum number of files per variant import job in the current AWS region
Analytics - Maximum shares per annotation store	Each supported Region: 10	<a href="#">Yes</a>	The maximum number of shares per annotation store in the current AWS region
Analytics - Maximum shares per variant store	Each supported Region: 10	<a href="#">Yes</a>	The maximum number of shares per variant store in the current AWS region
Analytics - Maximum size of each file in a variant import job	Each supported Region: 20 Gigabytes	<a href="#">Yes</a>	The maximum size of one file in a variant import job in the current AWS region
Analytics - Maximum size of each file in an annotation import job	Each supported Region: 20 Gigabytes	<a href="#">Yes</a>	The maximum size of one file in an annotation import job in the current AWS region
Analytics - Maximum variant stores	Each supported Region: 10	<a href="#">Yes</a>	The maximum number of variant stores in the current AWS region
Analytics - Maximum versions per annotation store	Each supported Region: 10	<a href="#">Yes</a>	The maximum number of versions per annotation store in the current AWS region

Name	Default	Adjustable	Description
Storage - Maximum concurrent read set activation jobs	Each supported Region: 25	<a href="#">Yes</a>	The maximum number of concurrent read set activation jobs in the current AWS region
Storage - Maximum concurrent sequence and reference store export jobs	Each supported Region: 5	<a href="#">Yes</a>	The maximum number of concurrent export jobs from a sequence or reference store in the current AWS region
Storage - Maximum concurrent sequence or reference store import jobs	Each supported Region: 5	<a href="#">Yes</a>	The maximum number of concurrent import jobs for a sequence or reference store in the current AWS region
Storage - Maximum read sets per activation job	Each supported Region: 20	<a href="#">Yes</a>	The maximum number of read sets per activation job in the current AWS region
Storage - Maximum read sets per export job	Each supported Region: 100	<a href="#">Yes</a>	The maximum number of read sets per export job in the current AWS region
Storage - Maximum read sets per import job	Each supported Region: 100	<a href="#">Yes</a>	The maximum number of read sets per import job in the current AWS region

Name	Default	Adjustable	Description
Storage - Maximum read sets per sequence store	Each supported Region: 1,000,000	<a href="#">Yes</a>	The maximum number of read sets in a sequence store in the current AWS region
Storage - Maximum reference stores	Each supported Region: 1	No	The maximum number of reference stores in the current AWS region
Storage - Maximum references per reference store	Each supported Region: 50	<a href="#">Yes</a>	The maximum number of references in a reference store in the current AWS region
Storage - Maximum sequence stores	Each supported Region: 20	<a href="#">Yes</a>	The maximum number of sequence stores in the current AWS region
Workflows - Maximum active GPUs	Each supported Region: 12	<a href="#">Yes</a>	The maximum number of concurrent active GPUs in the current AWS region
Workflows - Maximum active vCPUs	Each supported Region: 3,000	<a href="#">Yes</a>	The maximum number of concurrent active vCPUs in the current AWS region
Workflows - Maximum concurrent active runs	Each supported Region: 10	<a href="#">Yes</a>	The maximum number of active runs in the current AWS region

Name	Default	Adjustable	Description
Workflows - Maximum concurrent tasks per run	Each supported Region: 25	<a href="#">Yes</a>	The maximum number of concurrent tasks in each run in the current AWS region
Workflows - Maximum run duration	Each supported Region: 604,800 Seconds	<a href="#">Yes</a>	The maximum workflow run duration in the current AWS region
Workflows - Maximum run groups	Each supported Region: 1,000	<a href="#">Yes</a>	The maximum number of run groups in the current AWS region
Workflows - Maximum runs (active or inactive)	Each supported Region: 5,000	<a href="#">Yes</a>	The maximum number of runs (active or inactive) in the current AWS region
Workflows - Maximum shares per workflow	Each supported Region: 100	<a href="#">Yes</a>	The maximum number of shares per workflow in the current AWS region
Workflows - Maximum static run storage capacity per run	Each supported Region: 9,600	<a href="#">Yes</a>	The maximum static run storage capacity in gibibytes (GiB) for each run in the current AWS region
Workflows - Maximum workflows	Each supported Region: 1,000	<a href="#">Yes</a>	The maximum number of workflows in the current AWS region

## HealthOmics file size quotas

The following table shows the maximum supported values for storage files. These values aren't adjustable.

Name	Description	Maximum	Adjustable Yes/No
Storage - Maximum part size for a direct upload	The maximum part size for direct upload to a sequence store.	100 MB	No
Storage - Maximum parts in file for direct upload	The maximum number of parts in a file for direct upload to a sequence store in the current Region.	10,000	No
Storage - Maximum reference size	The maximum size of a reference file that can be imported to a reference store.	15 GB	No
Storage - Maximum read set source size	The maximum size of a single source file in a read set that can be imported to a sequence store.	976 GB	No

The following table shows the maximum supported sizes for workflow files. These values aren't adjustable.

Name	Description	Maximum size	Adjustable Yes/No
Workflows - Run parameter file	The maximum size of a run parameter file.	50,000 bytes	No



Each Ready2Run workflow has a maximum input file size. In the following table, the file size units are listed in Gibibytes (GiB). These maximum file sizes aren't adjustable.

Ready2Run workflow name	Maximum input file size (GiB)	Adjustable (Yes/No)
AlphaFold for 601-1200 residues	1	No
AlphaFold for up to 600 residues	1	No
Bases2Fastq for 2x150	1000	No
Bases2Fastq for 2x300	1000	No
Bases2Fastq for 2x75	500	No
ESMFold for up to 800 residues	1	No
GATK-BP fq2bam	64	No
GATK-BP Germline bam2vcf for 30x genome	39	No
GATK-BP Germline fq2vcf for 30x genome	64	No
GATK-BP Somatic WES bam2vcf	86	No
NVIDIA Parabricks BAM2FQ2BAM WGS for up to 30X	80	No
NVIDIA Parabricks BAM2FQ2BAM WGS for up to 50X	120	No

Ready2Run workflow name	Maximum input file size (GiB)	Adjustable (Yes/No)
NVIDIA Parabricks BAM2FQ2BAM WGS for up to 5X	20	No
NVIDIA Parabricks FQ2BAM WGS for up to 30X	71	No
NVIDIA Parabricks FQ2BAM WGS for up to 50X	137	No
NVIDIA Parabricks FQ2BAM WGS for up to 5X	13	No
NVIDIA Parabricks Germline DeepVariant WGS for up to 30X	71	No
NVIDIA Parabricks Germline DeepVariant WGS for up to 50X	137	No
NVIDIA Parabricks Germline DeepVariant WGS for up to 5X	12	No
NVIDIA Parabricks Germline HaplotypeCaller WGS for up to 30X	71	No
NVIDIA Parabricks Germline HaplotypeCaller WGS for up to 50X	137	No
NVIDIA Parabricks Germline HaplotypeCaller WGS for up to 5X	13	No

Ready2Run workflow name	Maximum input file size (GiB)	Adjustable (Yes/No)
NVIDIA Parabricks Somatic Mutect2 WGS for up to 50X	196	No
scRNAseq with KallistoB UStools	119	No
scRNAseq with Salmon Alevin-fry	119	No
scRNAseq with STARsolo	119	No
Sentieon Germline BAM WES for up to 300x	9	No
Sentieon Germline BAM WGS for up to 32x	18	No
Sentieon Germline FASTQ WES for up to 100x	5	No
Sentieon Germline FASTQ WES for up to 300x	26	No
Sentieon Germline FASTQ WGS for up to 32x	51	No
Sentieon LongRead for ONT	25	No
Sentieon LongRead for PacBio HiFi	58	No
Sentieon Somatic WES	50	No
Sentieon Somatic WGS	113	No
Ultima Genomics DeepVariant for up to 40x	91	No

## HealthOmics API quotas

HealthOmics has the following quotas related to API operations. Where indicated, the quota is adjustable. To request an increase, use the [quota increase form](#).

For each API operation listed, the quota is the maximum transactions per second (TPS) for that API operation in each Region.

The following table lists the storage API operations.

Storage API operation	Default maximum TPS	Adjustable Yes/No
CreateSequenceStore, CreateReferenceStore, DeleteSequenceStore, DeleteReferenceStore	1 TPS	Yes
BatchDeleteReadSet, DeleteReference	1 TPS	Yes
CreateMultipartReadSetUpload, CompleteMultipartReadSetUpload, AbortMultipartReadSetUpload	1 TPS	No
GetReference	10 TPS	Yes
UploadReadSetPart	10 TPS	Yes
GetReadSet	30 TPS	Yes
GetSequenceStore, ListSequenceStores	5 TPS	Yes
GetReadSetMetadata, ListReadSets	5 TPS	Yes
StartReadSetImportJob, GetReadSetImportJob, ListReadSetImportJobs	5 TPS	Yes

Storage API operation	Default maximum TPS	Adjustable Yes/No
StartReadSetExportJob, GetReadSetExportJob, ListReadSetExportJobs	5 TPS	Yes
ListReferenceStores	5 TPS	Yes
StartReferencetImportJob, GetReferenceImportJob, ListReferenceImportJobs	5 TPS	Yes
ListReferences, GetReferenceMetadata	5 TPS	Yes
StartReadsetActivationJob	5 TPS	Yes
ListReadsetActivationJobs, GetReadSetActivationJob	5 TPS	Yes
ListMultipartReadSetUploads, ListReadSetUploadParts	5 TPS	Yes
TagResource, UntagResource, ListTagsForResource	5 TPS	Yes

The following table lists the workflow API operations.

Workflow API operation	Default maximum TPS	Adjustable Yes/No
StartRun	0.1 TPS	Yes
CreateWorkflow	5 TPS	Yes
CancelRun, DeleteRun, , GetRun, GetRunTask, ListRunTasks, ListRuns	10 TPS	Yes

Workflow API operation	Default maximum TPS	Adjustable Yes/No
CreateRunGroup, DeleteRunGroup, GetRunGroup, ListRunGroups, UpdateRunGroup	10 TPS	Yes
DeleteWorkflow, GetWorkflow, ListWorkflows, UpdateWorkflow	10 TPS	Yes

The following table lists the analytics API operations.

Analytics API operation	Default maximum TPS	Adjustable Yes/No
CreateVariantStore, DeleteVariantStore, GetVariantStore, ListVariantStores, UpdateVariantStore	1 TPS	No
StartVariantImportJob, CancelVariantImportJob, GetVariantImportJob, ListVariantImportJobs	1 TPS	No
CreateAnnotationStore, DeleteAnnotationStore, GetAnnotationStore, ListAnnotationStores, UpdateAnnotationStore	1 TPS	No
StartAnnotationImportJob, ListAnnotationImportJobs, GetAnnotationImportJob, CancelAnnotationImportJob	1 TPS	No

# Document history for the HealthOmics User Guide

The following table describes the documentation releases for HealthOmics.

Change	Description	Date
<a href="#">New Features</a>	HealthOmics added support for shared workflows and dynamic run storage.	April 30, 2024
<a href="#">New Features</a>	HealthOmics added support for Amazon S3 access to reference and sequence stores, and support for SHA256 ETags.	April 15, 2024
<a href="#">New Features</a>	HealthOmics added entity tags (ETags) for sequence stores.	October 6, 2023
<a href="#">New Features</a>	HealthOmics added annotation store versioning and analytic store sharing.	August 15, 2023
<a href="#">New Features</a>	HealthOmics added Common Workflow Language (CWL) as a supported language for HealthOmics workflows.	June 30, 2023
<a href="#">New Features</a>	HealthOmics added new Ready2Run workflows, GPU support for workflows, data parsing for annotation stores, direct upload into HealthOmics storage, and integration with EventBridge.	May 15, 2023

[New managed policy](#)

HealthOmics added a new managed policy that provides full access. To learn more, see [AWS managed policies](#).

February 23, 2023

[New managed policy](#)

HealthOmics added a new managed policy that limits access to read only. To learn more, see [AWS managed policies](#).

November 29, 2022

[Initial release](#)

Initial release of the HealthOmics User Guide

November 29, 2022